

**A METHODOLOGY TO ACHIEVE MICROSCOPIC/MACROSCOPIC CONFIGURATION
TRADEOFFS IN COOPERATIVE MULTI-ROBOT SYSTEMS DESIGN**

Volume I



A Thesis
Presented to
The Academic Faculty

By

Jean-Guillaume Durand

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Aerospace Engineering

Georgia Institute of Technology
May 2017

Copyright © 2017 by Jean-Guillaume Durand

**A METHODOLOGY TO ACHIEVE MICROSCOPIC/MACROSCOPIC CONFIGURATION
TRADEOFFS IN COOPERATIVE MULTI-ROBOT SYSTEMS DESIGN**

Approved by:

Prof. Dimitri N. Mavris, Advisor
School of Aerospace Engineering
Georgia Institute of Technology

Prof. Daniel P. Schrage
School of Aerospace Engineering
Georgia Institute of Technology

Dr. Éric Feron
School of Aerospace Engineering
Georgia Institute of Technology

Dr. Jean-Marc Moschetta
Département Aérodynamique, Energétique et Propulsion
Institut Supérieur de l'Aéronautique et de l'Espace

Dr. K. Daniel Cooksey
School of Aerospace Engineering
Georgia Institute of Technology

Date Approved: February 9, 2017

Il n'y a qu'au pied du mur que tu vois si t'es aussi fort que tu crois.

Kery James

To my family
To Ludovic Brodut †

REMERCIEMENTS

À ce point tournant de mon éducation, il est de mon devoir de montrer ma gratitude envers les acteurs ayant rendu possible l'obtention de ce diplôme. Tout d'abord, mon maître de thèse Dr. Dimitri Mavris pour sa confiance dès mes premiers trimestres à Georgia Tech et son soutien répété au cours de ces quatre dernières années. Son expérience inégalable dans le milieu aéronautique est venue enrichir le présent travail de par ses nombreuses remarques perspicaces. Je tiens également à remercier mes autres encadrants de thèse pour avoir régulièrement revu mon travail et assuré sa qualité. Dr. Jean-Marc Moschetta et Dr. Éric Feron qui ont suivi mon parcours depuis mes années à l'Institut Supérieur de l'Aéronautique et de l'Espace (Supaero) et qui ont su apporter un point de vue extérieur à l'ASDL (Aerospace Systems Design Laboratory) ainsi qu'une perspective autre à la problématique principale de cette thèse. Dr. Schrage pour ses apports sur les concepts d'optimisation et de conception. Dr. K. Daniel Cooksey pour son suivi au jour le jour qui a permis de continuellement ajuster le cadre et les hypothèses de ce travail. Dr. Kelly Griendling pour avoir aidé à l'édification des bases fondatrices du projet de recherche.

Outre mes tuteurs de thèse, je tiens à remercier les quelques amis qui ont accompagné mon parcours. En particulier, Emmanuelle Charlot, pour avoir été mon soutien continu et mon réconfort dans les pénibles années précédant ma thèse. Telle une promesse restée immuable face aux auspices changeants, Nivedita Ravi, sans qui la présente thèse n'existerait pas, sans qui je ne me serais jamais embarqué dans l'aventure du doctorat. Frédéric Burgaud, pour son implication méticuleuse et ses nombreux conseils techniques lors des différentes étapes de cette thèse. Christopher Frank, pour ses recommandations et pour avoir été un exemple de rigueur et d'organisation sur la réalisation d'un doctorat. Houssam Assany, pour ces longues discussions tardives et enrichissantes sur les aspects techniques ainsi que les répercussions d'une telle entreprise. David Alléos, pour son soutien régulier au cours de ces quatre années loin de France. J'adresse également un grand merci à mes amis et autres camarades qui ont de près ou de loin, à un moment ou un autre, croisé mon chemin au cours de mes années à Georgia Tech.

Enfin, bien entendu et de manière plus générale, je remercie ma famille, proche et éloignée, comme elle a su au fil des années me soutenir de manière inconditionnelle dans ce projet académique tout comme dans les autres. Plus particulièrement mes parents, Isabelle Julienne Durand et Jean-François Durand, comme ce que je suis devenu n'est que le fruit et reflet de leur éducation. Ma sœur Mélodie et mon frère Lénaïc, pour m'avoir donné l'inspiration et la force de donner l'exemple du mieux que je pouvais. Une famille qui aura su faire de ses rêves une réalité, pour un enfant qui à l'époque dessinait des avions sur feuilles de papier.

ACKNOWLEDGMENTS

At this turning point of my education, it is my duty to express my gratitude to the protagonists who made this degree possible. First of all, my main thesis advisor Dr. Dimitri Mavris for his trust from my very first semesters at Georgia Tech and his repeated support during these past four years. His unmatched experience in the aerospace field enriched the present work with his many insightful remarks. I also really want to thank my other thesis supervisors for regularly reviewing my work and certifying its quality. Dr. Jean-Marc Moschetta and Dr. Éric Feron who have been following my journey since my years at the Institut Supérieur de l'Aéronautique et de l'Espace (Supaero), and who were able to bring an external point of view to the approach of the Aerospace Systems Design Laboratory (ASDL) and share another perspective to the main problematic of this thesis. Dr. Schrage for his contributions on optimization and design concepts. Dr. K. Daniel Cooksey for his day-to-day follow-up which allowed a continuous adjustment of the scope and assumptions of this work. Dr. Kelly Griendling for helping in establishing the groundwork of the thesis proposal.

Besides my supervisors, I want to acknowledge the few friends who accompanied my peregrination. In particular, Emmanuelle Charlot, for bringing continuous support and comfort during the distressful years preceding the thesis. As a promise remained immune to changing circumstances, Nivedita Ravi, without whom the present thesis would not be, without whom I would not have embarked in the Ph.D. adventure. Frédéric Burgaud, for his meticulous implication and his technical advice throughout the different stages of this thesis. Christopher Frank, for his guidance and for being an example of rigor and organization for the completion of a Ph.D. degree. Houssam Assany, for the late, long, and enriching discussions about technical aspects and repercussions of such an undertaking. David Alléos, for his regular support all through these four years away from France. I also would like to say a big thank you to my friends and classmates who, in one way or another, crossed my path during my years at Georgia Tech.

At last, naturally and in a more general way, I thank my family, immediate and distant, as it supported me unconditionally over the years in this academic project as well as in other endeavors. More importantly my parents Isabelle Julienne Durand et Jean-François Durand, since what I have become is none other than the fruit and the reflection of their education. My sister Mélodie and my brother Lénaïc, for giving me the strength and inspiration to lead by example the best I could. A family which translated dreams into reality for a kid who used to draw planes on pieces of paper.

TABLE OF CONTENTS

REMERCIEMENTS	v
ACKNOWLEDGMENTS	vi
LIST OF TABLES	xi
LIST OF FIGURES	xiv
LIST OF EQUATIONS.....	xxiii
LIST OF SYMBOLS AND ABBREVIATIONS	xxvii
SUMMARY	xxix
CHAPTER 1 Motivation	1
1.1 Brief overview of the research objective	2
1.2 The potential of unmanned systems.....	3
1.2.1 The advantages over human operators.....	4
1.2.2 A growing market	11
1.2.3 A fleet getting more diverse.....	19
1.2.4 The limitations	27
1.3 The growth of multi-robotics	37
1.3.1 A field inspired by nature	39
1.3.2 An increase in capability.....	48
1.3.3 Application to real-world problems	56
1.3.4 The limitations	58
1.4 Summary	64
1.4.1 Research objective	65

1.4.2	Research challenges	68
CHAPTER 2 Problem definition.....		74
2.1	Introductory example	75
2.2	Bridging the gap from microscopic to macroscopic level	85
2.2.1	Swarm engineering: a lack of maturity	86
2.2.2	A diversity of design methods	102
2.3	Exploring a large design space	124
2.3.1	Generating alternatives in a multi-architecture multi-level design space...	129
2.3.2	Optimizing in a multi-architecture multi-level design space	146
2.4	Summary	168
CHAPTER 3 Proposed approach.....		170
3.1	Establishment of performance metrics.....	171
3.1.1	Parallelism efficiency metrics.....	172
3.1.2	Introduction of marginal quantities.....	187
3.1.3	Benchmarking	195
3.2	Design space definition.....	200
3.2.1	The design variables	200
3.2.2	Alternatives generation	204
3.3	Alternatives evaluation	205
3.3.1	Microscopic level: the agents.....	207
3.3.2	Macroscopic level: the swarm	210
3.3.3	Agent-based simulation	213
3.3.4	Testbed mission	219

3.4	Decision-making process	224
3.5	Verification and validation	225
3.6	Summary	226
CHAPTER 4 Linking microscopic and macroscopic levels		232
4.1	An improvement for the design of multi-robot systems	232
4.1.1	Global optimization algorithm	236
4.1.2	Sequential optimization algorithm	236
4.1.3	Verification and validation	240
4.1.4	Experimentation	240
4.1.5	Conclusions	257
4.2	Mesosopic modeling	259
4.2.1	Canonical mission	260
4.2.2	Macroscopic model	265
4.2.3	Microscopic model	266
4.2.4	Mesosopic model	274
4.2.5	Verification and validation	280
4.2.6	Characterization	281
CHAPTER 5 Multi-architecture multi-level design space exploration		291
5.1	Generation of alternatives: the tree of reduced morphological matrices	292
5.1.1	Step 1: morphological reduction	293
5.1.2	Step 2: morphological tree	298
5.1.3	Implementation	301
5.1.4	Verification and validation	323

5.1.5	Characterization	325
5.2	Design optimization: the bi-level genetic algorithm.....	345
5.2.1	Implementation	346
5.2.2	Verification and validation	364
5.2.3	Characterization	376
CHAPTER 6 Conclusion.....		394
6.1	Research summary	394
6.2	Closing the loop: MASDeM	399
6.3	Key contributions.....	403
6.4	Perspectives of future research	407
APPENDIX.....		410
REFERENCES.....		642
VITA.....		666

LIST OF TABLES

Table 1.1: Manned vs. unmanned mission cost comparisons [32]	11
Table 1.2: Swarm robotics taxonomy axes [101]	44
Table 2.1: Example 1 designs	83
Table 2.2: Example 2 designs	85
Table 2.3: Overview of multi-robot simulators	89
Table 2.4: Design methods review.....	118
Table 2.5: Analysis methods review	119
Table 2.6: UAV capabilities by category.....	131
Table 2.7: Review of alternatives generation methods.....	137
Table 2.8: Notional UAV morphological matrix.....	140
Table 2.9: Enhanced morphological matrix.....	140
Table 2.10: Review of multi-objective optimization techniques	156
Table 2.11: Comparison of MDO frameworks [149], [195], [199].....	162
Table 3.1: Marginal quantities for the introductory example	191
Table 3.2: Marginal quantities over complete design space	193
Table 3.3: A disparity of metrics	196
Table 3.4: A set of unified metrics.....	199
Table 3.5: Notional morphological matrix at the macroscopic level.....	204
Table 3.6: Swarm control architectures	212
Table 3.7: Agent-based simulation platforms comparison	218
Table 4.1: Results of experiment 1.1	243

Table 4.2: Results of experiment 1.2	245
Table 4.3: Results of experiment 1.3	247
Table 4.4: Results of experiment 1.4	249
Table 4.5: Results of experiment 1.6	253
Table 5.1: One option chosen per feature	295
Table 5.2: Grouping of alternatives into architectures.....	298
Table 5.3: Example of empennage morphological matrix.....	307
Table 5.4: Fixed-wing architecture morphological interface.....	308
Table 5.5: Multicopter architecture morphological interface	309
Table 5.6: Macroscopic level morphological interface.....	310
Table 5.7: Morphological interface for the airship architecture	311
Table 5.8: Morphological interface for the ornithopter architecture [232], [233]	312
Table 5.9: Options compatibility indexing convention.....	313
Table 5.10: Number of compatible alternatives.....	318
Table 5.11: Number of reduced alternatives.....	320
Table 5.12: Average number of function calls for the test functions.....	327
Table 5.13: Outer loop test function values	366
Table 5.14: Improved convergence with elitism.....	382
Table 5.15: Main effect of elitism rate.....	384
Table 5.16: Optimizer characterization for stall-based convergence.....	392
Table 5.17: Optimizer characterization for precision-based convergence.....	393
Table 6.1: Summary of research questions and hypotheses	395

Table A-1: KKT analysis for inner loop constraints.....	426
Table A-2: KKT analysis for outer loop constraints.....	430
Table A-3: Optimum of the Michalewicz test function	436

LIST OF FIGURES

Figure 1.1: Unmanned aerial vehicles used in military operations [1]	3
Figure 1.2: Cost of main DoD programs [6, 7, 8, 9, 10, 11, 12, 13, 14, 15], [16, 17, 18, 19, 20, 21]	6
Figure 1.3: Example of aerial imagery solutions	7
Figure 1.4: USAF aircraft cost per flight hour [27]	9
Figure 1.5: Imaging cost per platform [31].....	10
Figure 1.6: UAV global market forecast [33, 34, 35, 36].....	12
Figure 1.7: An example of drone airspace integration [38]	13
Figure 1.8: Total U.S. UAS Systems in the National Airspace System [33, 39].....	14
Figure 1.9: Expected military UAV revenues (Europe) [40].....	15
Figure 1.10: Drone programs from the giant tech companies	16
Figure 1.11: Parrot A.R. Drone 2.0 price evolution [47, 48]	17
Figure 1.12: Price reduction for Phantom 2 variants [49, 50]	18
Figure 1.13: Civil micro quadcopter UAV market [53]	20
Figure 1.14: Examples of classical and unconventional designs	22
Figure 1.15: Interface of ArduPilot [65, 66]	23
Figure 1.16: Notional family of reconfigurable aircraft [67].....	25
Figure 1.17: Use of 3D printing on a quadcopter design [69]	26
Figure 1.18: Use of 3D printing on a fixed-wing design [70]	26
Figure 1.19: Equivalent DJI Phantom battery life over time [72, 73, 74]	28
Figure 1.20: Network settings of the Parrot Bebop drone [78].....	31

Figure 1.21: Taking advantage of fleet diversity [83, 84]	34
Figure 1.22: Total personnel for drone combat air patrols [86, 87, 88, 89, 90].....	35
Figure 1.23: Approval of drone use to kill high-level terrorism suspects overseas [91] ..	36
Figure 1.24: Example of robotic swarm [94].....	38
Figure 1.25: Swarm behaviors in nature	40
Figure 1.26: Swarm behavior rules in Particle Swarm Optimization [106]	45
Figure 1.27: Holes avoidance through group reconfiguration [112]	49
Figure 1.28: Chain formation for a narrow passage [112].....	50
Figure 1.29: Five robots collectively tackle a 14cm step [112].....	50
Figure 1.30: Cooperative transport using quadrotors	51
Figure 1.31: Area surveillance by UAVs [115]	52
Figure 1.32: Multiple observations of a map landmark	53
Figure 1.33: Bridge construction using distributed action [116]	54
Figure 1.34: Representation of a UAV-based search and rescue operation [119].....	57
Figure 1.35: Representation of a multi-UAS military system [117].....	58
Figure 1.36: Establishment of the research objective	67
Figure 1.37: An extremely large design space.....	70
Figure 1.38: Prevalence of cost considerations in early design phases	72
Figure 2.1: Decomposition of the research objective	74
Figure 2.2: Example of mapping configuration for 3 agents	76
Figure 2.3: Distance back to initial mapping point.....	78
Figure 2.4: Evolution of mapping time with the design variables (example 1).....	81
Figure 2.5: Contours for mapping rate and system cost (example 1)	82

Figure 2.6: Evolution of mapping time with the design variables (example 2).....	84
Figure 2.7: Missing link between macroscopic and microscopic level	90
Figure 2.8: Example of sequential swarm design optimization	94
Figure 2.9: Example of global swarm design optimization	94
Figure 2.10: Wing design by sequential disciplinary optimization	96
Figure 2.11: Optimization in supply chain management	98
Figure 2.12: Swarm engineering methods taxonomy [95].....	102
Figure 2.13: Architecture-based SoS engineering process	106
Figure 2.14: Probabilistic finite state machine design	108
Figure 2.15: Virtual physics-based design.....	109
Figure 2.16: Capabilities of current design space exploration techniques.....	126
Figure 2.17: [173] proposed approach	127
Figure 2.18: Proposed approach	129
Figure 2.19: A capability-based taxonomy of UAVs [178].....	130
Figure 2.20: Force-field analysis	133
Figure 2.21: Morphological matrix reduction.....	142
Figure 2.22: Example of morphological matrix tree.....	143
Figure 2.23: Proposed alternatives generation method.....	144
Figure 2.24: Partial ordering and Pareto frontier sampling	153
Figure 2.25: Proposed optimization scheme	166
Figure 2.26: Summary of the problem definition process	169
Figure 3.1: Generic top-down design decision support process	170
Figure 3.2: Amdahl's law	175

Figure 3.3: Gustafson’s law compared to Amdahl’s law	177
Figure 3.4: Heterogeneous parallelization	179
Figure 3.5: Limits of parallelization for the introductory example	181
Figure 3.6: Controlled systems characteristic times	183
Figure 3.7: Visualization of LOPE for the introductory example	185
Figure 3.8: Detail of LOPE at 5% for the introductory example.....	185
Figure 3.9: All LOPE quantities for the introductory example	186
Figure 3.10: Sensitivity analysis around optimum design	187
Figure 3.11: A taxonomy of existing UAVs.....	202
Figure 3.12: Review of hypothesis 3.1	205
Figure 3.13: Agent modeling breakdown	207
Figure 3.14: Proposed agent-based model architecture	214
Figure 3.15: Representation of the testbed mission.....	219
Figure 3.16: Aerial imagery [218]	221
Figure 3.17: Orthomosaics [218].....	221
Figure 3.18: Common geometric transformations	222
Figure 3.19: Ortho-rectification process	222
Figure 3.20: Mosaicking [222]	223
Figure 3.21: Surface models [218].....	223
Figure 3.22: Summary of the research process	227
Figure 3.23: Proposed design space exploration technique summary	228
Figure 4.1: Sequential optimization without correlation	234
Figure 4.2: Sequential optimization with correlation	235

Figure 4.3: Micro-macro optimizer.....	237
Figure 4.4: Macro-micro optimizer.....	238
Figure 4.5: Plot of experiment 1.1	242
Figure 4.6: Plot of experiment 1.2	245
Figure 4.7: Plot of experiment 1.3	247
Figure 4.8: Plot of experiment 1.4	249
Figure 4.9: Plot of experiment 1.5	252
Figure 4.10: Plot of experiment 1.6	253
Figure 4.11: 3D visualization of principal component analysis	254
Figure 4.12: 2D representation of principal component analysis	255
Figure 4.13: Heterogeneous canonical example	256
Figure 4.14: Robotarium project logo.....	260
Figure 4.15: Robotarium testbed.....	261
Figure 4.16: GRITBots robot platform	262
Figure 4.17: Rendezvous trajectories on Robotarium implementation [226].....	264
Figure 4.18: PR2 robot model.....	267
Figure 4.19: CAD model of the GRITBots.....	268
Figure 4.20: Simulated GRITBots model	269
Figure 4.21: Microscopic model architecture	271
Figure 4.22: Pose tracking system simulation	271
Figure 4.23: Robotarium arena and GRITBots in the Gazebo simulator	273
Figure 4.24: Different levels of detail for aircraft models	274
Figure 4.25: Unicycle model representation.....	276

Figure 4.26: Graphic interface of the mesoscopic model	280
Figure 4.27: The increasing level of detail of the implemented GRITBot models.....	281
Figure 4.28: Consensus time metric.....	282
Figure 4.29: Section view of consensus time response for $v = 1$ cm/s	283
Figure 4.30: Robot trajectories for $N=3$ and $v=8$ cm/s	284
Figure 4.31: Precision of the models on the consensus position metric	285
Figure 4.32: Precision of the models for the consensus time metric	286
Figure 4.33: Average runtime of the different models	288
Figure 4.34: Notional representation of the precision of mesoscopic modeling	290
Figure 5.1: Example of compatibility matrix.....	294
Figure 5.2: Number of filled elements in compatibility matrix	296
Figure 5.3: Regrouping of options	297
Figure 5.4: Example of morphological tree	300
Figure 5.5: Multi-level morphological tree.....	301
Figure 5.6: Proposed UML class diagram for the morphological tree	303
Figure 5.7: Compatibility matrices	314
Figure 5.8: Compatibility matrices pattern	315
Figure 5.9: Recursive count of compatible alternatives.....	317
Figure 5.10: Problem conditioning	319
Figure 5.11: Effect of options removal on overall reduction.....	321
Figure 5.12: Initial morphological tree before reduction.....	322
Figure 5.13: Number of function calls versus number of design variables	326
Figure 5.14: General effect of morphological reduction	328

Figure 5.15: Limits of morphological reduction.....	330
Figure 5.16: Influence of factor k on morphological reduction.....	331
Figure 5.17: Contours of morphological reduction profitability	332
Figure 5.18: Influence of the number of rows on morphological reduction.....	333
Figure 5.19: Influence of the number of options per row on morphological reduction..	334
Figure 5.20: Strategies of options removal.....	335
Figure 5.21: Influence of options removal strategy on morphological reduction.....	336
Figure 5.22: Notional morphological reduction on a bi-level problem	338
Figure 5.23: Relative morphological reduction	339
Figure 5.24: Influence of problem size on bi-level morphological reduction	340
Figure 5.25: Complete morphological reduction on bi-level problem	341
Figure 5.26: Influence of both levels on morphological reduction.....	342
Figure 5.27: Variables profiler for morphological reduction.....	343
Figure 5.28: High-level architecture of the proposed bi-level optimizer.....	346
Figure 5.29: Dynamic size allocation for inner loop chromosomes	347
Figure 5.30: Dynamic size allocation formulae.....	348
Figure 5.31: Population initialization at every instantiation.....	350
Figure 5.32: Population initialization at every generation.....	351
Figure 5.33: Elite retention with elite memory.....	353
Figure 5.34: Elite retention with buffer only	354
Figure 5.35: Full vs. partial heterogeneity	356
Figure 5.36: Genetic algorithm vs. full factorial.....	359
Figure 5.37: Simplified expression for full factorial approach.....	361

Figure 5.38: Choosing between full factorial or genetic algorithm	362
Figure 5.39: Outer loop test function offsets	366
Figure 5.40: Design variables nomenclature	368
Figure 5.41: Variables ordering in the inner loop design vectors.....	369
Figure 5.42: Inner loop test function visualization	370
Figure 5.43: Inner loop test function sectional cut	371
Figure 5.44: Effect of elitism	377
Figure 5.45: Effect of elitism rate	380
Figure 5.46: Effects of elitism on the outer loop	382
Figure 5.47: Detail on effect of elitism.....	383
Figure 5.48: Initial effects of elitism.....	384
Figure 5.49: Effect of partial heterogeneity and elitism	386
Figure 5.50: Effect of heterogeneity type on stall-based outer loop convergence.....	387
Figure 5.51: Effect of heterogeneity type on stall-based inner loop convergence.....	388
Figure 5.52: Effect of heterogeneity type on precision-based outer loop convergence..	390
Figure 5.53: Increased convergence instability with partial heterogeneity	391
Figure 6.1: Summary of the research process	398
Figure 6.2: Steps of MASDeM	400
Figure A-1: Constraints on the first design variable	427
Figure A-2: Constraints on the second design variable	428
Figure A-3: Constraints on two design variables.....	428
Figure A-4: Ackley function representation	432

Figure A-5: Dixon-Price function representation	433
Figure A-6: Griewank function representation over a varied range	434
Figure A-7: Levy function representation.....	435
Figure A-8: Michalewicz function representation	437
Figure A-9: Local representation of the Michalewicz function.....	437
Figure A-10: Rastrigin function representation	438
Figure A-11: Rosenbrock function representation.....	439
Figure A-12: Rotated Hyper-Ellipsoid function representation.....	440
Figure A-13: Schwefel function representation	441
Figure A-14: Sphere function representation.....	442
Figure A-15: Styblinski-Tang function representation	443
Figure A-16: Sum of squares function representation	444

LIST OF EQUATIONS

Equation 1.1: n multi-choose k	69
Equation 1.2: Non-linearity of the design space	71
Equation 2.1: Total mission time	77
Equation 2.2: Mapping time for each agent.....	77
Equation 2.3: Distance from base to mapping area	77
Equation 2.4: Distance traveled during mapping phase.....	78
Equation 2.5: Distance to return to the base	79
Equation 2.6: Total distance	79
Equation 2.7: Final expression for total mapping time.....	80
Equation 2.8: Cost structure.....	80
Equation 2.9: Optimization problem formulation.....	148
Equation 2.10: Linear aggregate function.....	149
Equation 2.11: Nonlinear aggregate function	150
Equation 2.12	151
Equation 2.13: Overall evaluation criterion.....	152
Equation 2.14: Weighted p-norm of the objectives	154
Equation 3.1: Speedup formula.....	173
Equation 3.2: Parallelism efficiency	173
Equation 3.3: Parallelized execution time	173
Equation 3.4: Amdahl's law derivation.....	174
Equation 3.5: Parallel efficiency with Amdahl's law	174

Equation 3.6: Workload-based speedup formula.....	175
Equation 3.7: Parallelized workload.....	176
Equation 3.8: Gustafson’s law derivation.....	176
Equation 3.9: Heterogeneous Amdahl’s law speedup	178
Equation 3.10: Heterogeneous Gustafson’s law speedup.....	178
Equation 3.11: Solution of the parallelization toy example.....	179
Equation 3.12: First order system equation	183
Equation 3.13: Marginal Group Performance.....	189
Equation 3.14: Numerality Marginal Group Performance	189
Equation 3.15: Marginal Group Cost.....	190
Equation 3.16: Marginal Group Efficiency	190
Equation 3.17: Execution index.....	197
Equation 3.18: Completion index	198
Equation 3.19: Cost index.....	198
Equation 4.1: Uncorrelated example function	234
Equation 4.2: Correlated example function	235
Equation 4.3: Mathematical formulation of the rendezvous problem	263
Equation 4.4: First metric formula for the macroscopic model.....	265
Equation 4.5: Second metric formula for the macroscopic model.....	266
Equation 4.6: Laplacian matrix for N=5 agents.....	272
Equation 4.7: Kinematic model of a unicycle robot	276
Equation 4.8: Dynamics of the Gritbots controller.....	277
Equation 4.9: Inverse control commands for the Gritbots.....	278

Equation 4.10: System identification model.....	278
Equation 4.11: Final values for system identification	279
Equation 5.1: Number of alternatives from morphological analysis	294
Equation 5.2: Number of filled elements in compatibility matrix	296
Equation 5.3: Total number of alternatives for a combinatorial row.....	305
Equation 5.4: Surrogate model for the number of function calls.....	327
Equation 5.5: General equation for bi-level morphological reduction	342
Equation 5.6: Overall numerality constraints	354
Equation 5.7: Individual numerality constraints.....	355
Equation 5.8: Outer loop optimization problem	355
Equation 5.9: Inner loop optimization problem.....	355
Equation 5.10: Outer loop test function.....	365
Equation 5.11: Inner loop test function.....	367
Equation 5.12: Optimal value for design variable i	368
Equation 5.13: Optimal value for the test function variables	369
Equation 5.14: Test function for the complete algorithm.....	372
Equation 5.15: Constrained outer loop optimization problem.....	375
Equation 5.16: Constrained inner loop optimization problem.....	375
Equation A-1: Unconstrained verification function for the optimizer.....	416
Equation A-2: Gradient of the verification function.....	418
Equation A-3: Hessian definition.....	419
Equation A-4: Hessian matrix for the verification function	420

Equation A-5: Hessian matrix evaluated at the stationary point.....	420
Equation A-6: Unconstrained global optimum	421
Equation A-7: Minimum value of the test function at the unconstrained optimum	421
Equation A-8: Constrained outer loop optimization problem.....	422
Equation A-9: Constrained inner loop optimization problem.....	422
Equation A-10: Constrained optimization problem	423
Equation A-11: Standard form for the constrained optimization problem	423
Equation A-12: Lagrangian of the verification optimization problem	424
Equation A-13: Lagrangian derivative with respect to the inner loop design vector	424
Equation A-14: Lagrangian derivative with respect to the inner loop design vector	425
Equation A-15: KKT conditions for inner loop constraints.....	425
Equation A-16: Derivation of the inner loop constrained optimum	427
Equation A-17: KKT conditions for the outer loop	429
Equation A-18: Matricial form of inter-robot collision constraints.....	446
Equation A-19: World boundaries constraints.....	447
Equation A-20: Matricial form of world boundaries constraints.....	447
Equation A-21: Quadratic program based controller	448

LIST OF SYMBOLS AND ABBREVIATIONS

AADL	Architecture Analysis and Design Language (AADL)
AI	Artificial Intelligence
AMM	Augmented Morphological Matrix
ASDL	Aerospace Systems Design Laboratory
BLISS	Bi-Level Integrated System Synthesis
CFD	Computational Flow Dynamics
DAI	Distributed Artificial Intelligence
DoD	Department of Defense
DoE	Design of Experiments
DOF	Degrees of Freedom
FAA	Federal Aviation Administration
GSD	Ground Sampling Distance
GUST	Georgia Tech UAV Simulation Tool
HALE	High Altitude Long Endurance
HTA	Heavier Than Air
IOS	Individual, Organizational, and Societal
IRMA	Interactive Reconfigurable Matrix of Alternatives
JTRS	Joint Tactical Radio System
LiPo	Lithium Polymer
LOCUST	Low-Cost UAV Swarming Technology
LOPE	Limit Of Parallel Effectiveness
LTA	Lighter Than Air
LTE	Long-Term Evolution
NASA	National Aeronautics and Space Administration
NBI	Normal Boundary Intersection
NED	North-East-Down (frame)
NNC	Normalized Normal Constraint

MALE	Medium Altitude Long Endurance
MASDeM	Multi-Agent Systems Design Methodology
MAV	Micro Air Vehicle
MDO	Multidisciplinary Design Optimization
MGC	Marginal Group Cost
MGE	Marginal Group Efficiency
MGP	Marginal Group Performance
MUAV	Micro Unmanned Aerial Vehicle
OEC	Overall Evaluation Criterion
ONR	Office of Naval Research
PFSM	Probabilistic Finite State Machine
PID	Proportional–Integral–Derivative
PSO	Particle Swarm Optimization
R&D	Research and Development
RC	Radio-Controlled
SoS	System of Systems
SysML	Systems Modeling Language
TIES	Technology Identification Evaluation Selection
TIF	Technology Impact Forecasting
TOPSIS	Technique for Order Preference by Similarity to Ideal Solution
TRIZ	Theory of Inventive Problem Solving
TUAV	Tactical Unmanned Aerial Vehicle
UAS	Unmanned Aircraft System
UAV	Unmanned Aerial Vehicle
URDF	Unified Robot Description Format
US	United States (of America)
VEGA	Vector Evaluated Genetic Algorithm

SUMMARY

The exponential growth experienced by the robotics sector over the past decade has fostered the proliferation of new architectures. Optimized for specific missions, these platforms are in most cases limited by their embarked computational power and a lack of full situational awareness. More robust, flexible, scalable, and inspired by nature, group robotics represent an interesting approach to overcome some limitations of these single agents and take advantage of the heterogeneity of the current robotics fleet. Their essence lies in accomplishing more complex synergistic behaviors through diversity, simple rules, and local interactions. However, the design of robotic groups is complex as decision-makers have to optimize the group operation as well as the performance of each individual unit, for the group performance. In particular, key questions arise to know whether resources should be allocated to the characteristics of the group, or to the individual capabilities of its agents in order to meet the established requirements.

Current methods of swarm engineering tend to perform sequential optimization of the microscopic level (the agents) and then the macroscopic level (the group), which results in suboptimal architectures. In this context, efficiently comparing two different groups or quantifying the superiority of a group versus a single-robot design may prove impossible. Same goes of the determination of an optimal architecture for a given mission. With a special emphasis on aerial vehicles, the present research proposes to establish a methodology to achieve microscopic/macroscopic configuration tradeoffs in the design of cooperative multi-robot systems.

The resulting product is the MASDeM: Multi-Agent Systems Design Methodology. A novel multi-level multi-architecture morphological approach is first introduced to facilitate design space exploration, and a mesoscopic level simulation-based design method is used to bridge the gap between microscopic and macroscopic levels. Using these first blocks, an innovative optimization technique is suggested based on two interconnected loops which differs from the classical sequential approach presently used by the research community.

Results of this research show that simultaneous optimization can have clear benefits if applied to the design of multi-robot systems and on particular cases, average improvements of 16 percent were observed on the main performance metric. The proposed optimizer proves to be a key enabler for fully heterogeneous swarms, a capability which is not possible in the current paradigm. Moreover, the optimization algorithm was efficiently designed and exhibits a speedup of at least 50 percent when compared to current techniques. Finally, the exploration of the design space is effectively carried out with a combination of morphological reduction, morphological tree representation, and mesoscopic modeling. Indeed, applied to multi-robot systems, such models prove being several times faster than usual simulation approaches while remaining in the same range of accuracy.

This work is divided into two volumes with the appendix detailed in the second volume.

Keywords: Conceptual design, Multi-robotics, Swarm engineering, Mesoscopic, Design Optimization

CHAPTER 1

MOTIVATION

Automation is part of the quest for comfort of human beings and enables to autonomously carry out processes with minimal human intervention. It has been experiencing a relentless growth from even before the invention of the printing press by Gutenberg, the Jacquard loom or the centrifugal governor of Watt, to modern autonomous robots. The development of the robotics field unleashes a new potential for the automation of jobs that were only accomplished by humans so far. In particular, mobile unmanned systems provide advantages over human operators in many tasks including transportation of goods and people, delivery, or surveillance missions. This spectrum of robot operations is getting wider and wider as the current fleet is getting more diverse in terms of architectures and capabilities. Nonetheless, individual robots experience several limitations, some of which can be addressed through swarming. Directly inspired by nature, multi-robotics solutions such as robotic swarms propose increased capabilities over individual agents and enable to capitalize on the heterogeneity of the current fleet of robots. However, designing such systems of systems is a challenging task and the advantages of multi-robot systems over single-robot solutions need further examination. Their democratization remains impeded by the lack of a standard design process, delaying the use of multi-robot systems in industrial applications. This first section studies in greater detail these elements, drawing attention to certain needs leading to the research objective of this work.

1.1 Brief overview of the research objective

In order to ease the reader into understanding this research, the first chapter builds up on a series of observations in the field of multi-robot systems design:

- A growing diversity of drone types is now available and multi-robot collaboration may overcome the limitations of single robot platforms.
- Designing a multi-robot system requires much more commitment than for a single agent, such systems also tend to be confined to experimental applications.
- Multi-agent systems do not always perform “better” than single agents, most of the community focuses on homogeneous and sub-optimal group configurations.

A set of corresponding complementary assertions is then deduced from these observations:

- There is a potential to take advantage of the diversity of the existing drone fleet.
- A standard physical design methodology is required for multi-robot systems.
- There is a need to compare the performance of optimized multi-robot systems versus optimized single-robot platforms for a given mission.

Finally, these averments prompt the formulation of a unified research objective: the fundamental problematic engendering this research. Hence, the goal will be to establish a design methodology that enables the optimal design of multi-robot systems. In particular, key trade-offs will be examined such as the compromise between the number of agents in a group, and the individual capabilities of each agent of this group. The rest of this chapter goes on to detail, assemble, and give ground to the motivation of this work on the MASDeM: the Multi-Agent Systems Design Methodology.

1.2 The potential of unmanned systems

Thanks to dramatic improvements in computational power, battery life, miniaturization, and complexity of sensors, unmanned systems are now an essential constituent of military instrumentation and are finding their way into commercial and civil applications. What were once considered as cumbersome vehicles are nowadays proving as necessary assets for applications that were unforeseen a few years ago. Unmanned systems have now become an integral part of military operations as the recent airstrikes campaign in Syria and Afghanistan demonstrates [1].



Figure 1.1: Unmanned aerial vehicles used in military operations [1]

They are also being used more and more in commercial applications for mine mapping, building inspection, crops monitoring, etc. This section examines what the advantages proposed by unmanned systems are, as well as what their limitations consist in, despite a growing heterogeneous market.

1.2.1 The advantages over human operators

The use of unmanned systems is justified as they provide advantages over human operators and many of these benefits stem from the origins of automation. Indeed, the first advantages of unmanned systems are to be able to carry out tasks that would be too dangerous, complex, repetitive, or strenuous for human beings [2]:

- Dangerous operations include exploration of damaged buildings, mine clearance, missions in hostile sea storms, or miscellaneous jobs in radioactive environments. The hardware used in robots is more robust and resistant to environmental conditions that could be threatening for human life. In particular, they are able to withstand greater wear, shocks, and more extremes temperatures. For instance, the design of an Unmanned Air Vehicle (UAV) does not account for the accelerations and maneuverability constraints imposed by the human pilot on other manned vehicles. Sometimes, the danger lies only in reaching a remote area such as a collapsed building where small and aerial vehicles alone, more mobile than human workers, can penetrate. The fact that robots are an expendable asset also enables to send them on hazardous missions instead of humans [3].
- Complex tasks can consist of dense and accurate 3D mapping of a building or precise parts assembly. Owing to their elaborate sensors such as cameras, laser range finders, or else sonars, unmanned systems possess a much greater precision in sensual perception than humans. This enhanced perception is critical for tasks requiring accuracy.
- A typical repetitive process often carried out by unmanned systems nowadays is surveillance of buildings or borders. Automation enables to encode the repetitive

sequences in the logic of the robot only once. To take care of other strenuous tasks, powerful actuators make robots stronger than their human designers, and unmanned systems are capable of carrying out demanding operations such as lifting heavy payloads.

Thanks to the enhanced precision in terms of both perception and actuation, efficient control schemes can be implemented on the platforms, enabling a greater sensitivity in the required operations than with human operatives. This results in an unerring robustness in tasks completion or in achieving quality standards. Consequently, this consistency enables savings in wasted energy and materials, and the tasks are accomplished with improvements in quality and precision when compared with human workers. Additional savings in energy can be achieved by optimizing the behavior of the robots. For instance, a current motivation mentioned by [4] and [5] is to design movement patterns necessitating less acceleration energy. The controls are developed to calculate the best trajectories for the robots from the standpoint of energy efficiency. For mobile robots, this translates into an increase in operational range and endurance.

Additional advantages of unmanned systems over humans are cost related. In terms of mobile robotics, they generally present a cheaper acquisition cost than their manned counterparts despite the cost of automation and communication. The example of the military can first be considered as it is the breeding ground for the democratization of unmanned systems. Figure 1.2 presents the flyaway cost for the major defense acquisition programs defined by the Department of Defense (DoD) of the United States of America.

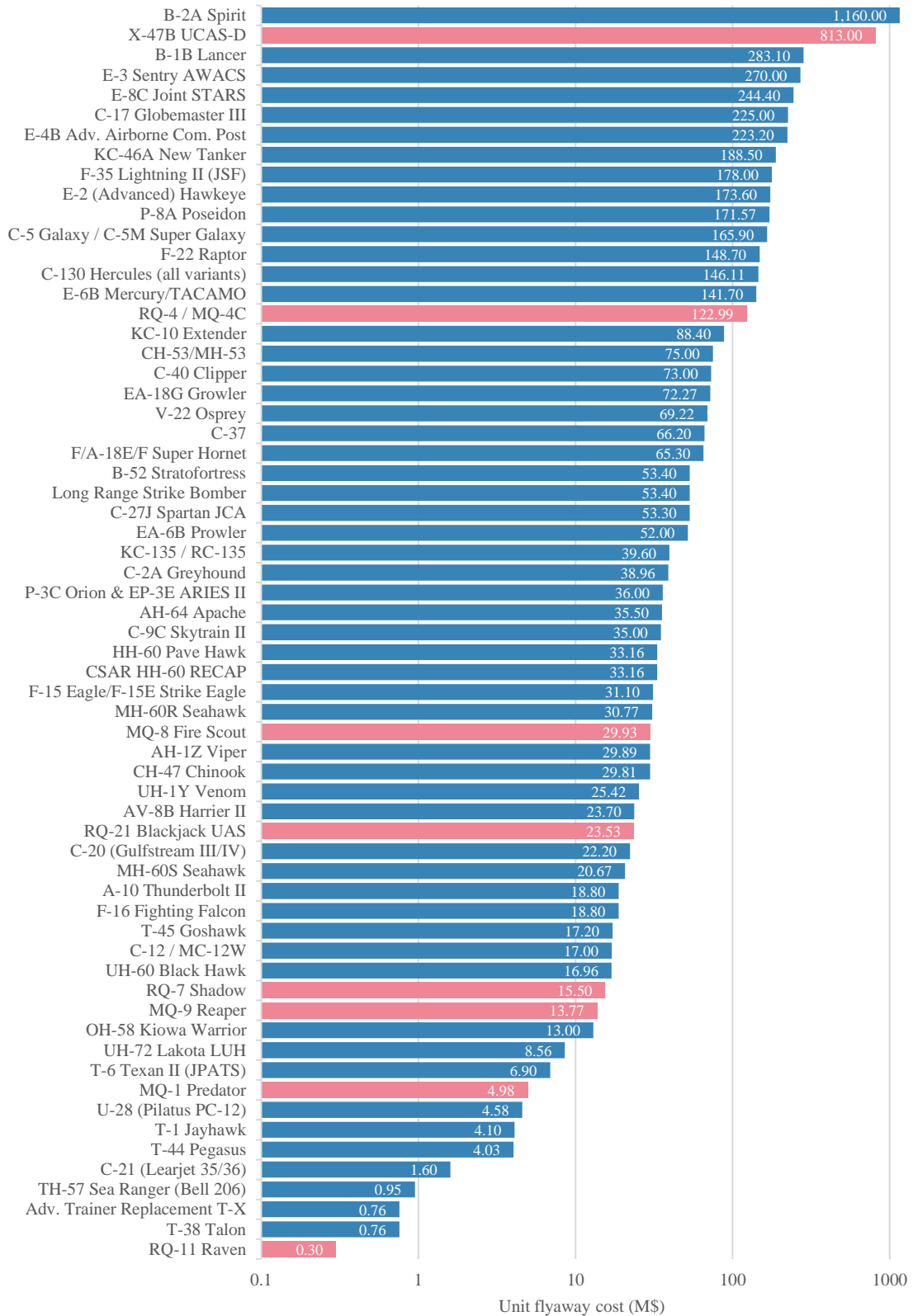


Figure 1.2: Cost of main DoD programs [6, 7, 8, 9, 10, 11, 12, 13, 14, 15], [16, 17, 18, 19, 20, 21]

While UAS costs span the same spectrum as their manned counterparts, it should be noted that the X-47 is a demonstrator only and that most unmanned systems tend to be in the lower part of the graph. Also, considering the versatile missions of the diverse aircraft presented on Figure 1.2, unmanned systems represent a less expensive option for fixed and pre-established requirements. This is especially true for surveillance and reconnaissance missions. These previous remarks highlight that unmanned systems are generally cheaper alternatives than manned systems in terms of acquisition costs.

Focusing now on civilian applications and the example of aerial imagery, the acquisition cost of the Landsat 8 imaging satellite is estimated at around \$855 million (including launch and operation) [22]. A Cessna 172 used for the same purpose would cost around \$300,000 while an automated mapping drone such as the senseFly's eBee RTK costs about \$25,000 and a simple imaging drone like the popular DJI's Phantom 3 is listed at \$1,000 (see Figure 1.3). As in the military, and although the satellite remains an unmanned system, the difference is clear in terms of acquisition cost for small unmanned air vehicles.



Figure 1.3: Example of aerial imagery solutions

While this latter is reduced by preferring an unmanned system option to a human-based one, cost savings are also encountered in utilization. Indeed, the savings in energy and materials mentioned hereinabove can lower the utilization cost for a given task or mission. Estimating the operational cost on an hourly basis is a difficult task when it comes to comparing unmanned and manned operations. Their mission profiles can be quite different as they generally do not fly at the same altitudes due to regulations or safety factors. The capabilities of one or the other also affect the type of weather and time conditions they can operate in. Moreover, the logistics involved in the transportation of the systems are different since one can be easily driven to a location while manned systems most likely have to be flown to the region of the mission. Unmanned systems tend to require less maintenance since their equipment is commercial-off-the-shelf and is subject to less safety requirements. However, the data gathered by UAS requires post-processing steps while a pilot can visually process information during the flight: such phases incur additional costs.

Emphasizing on the military applications first, many examples of cost per flight hour are presented in Figure 1.4.

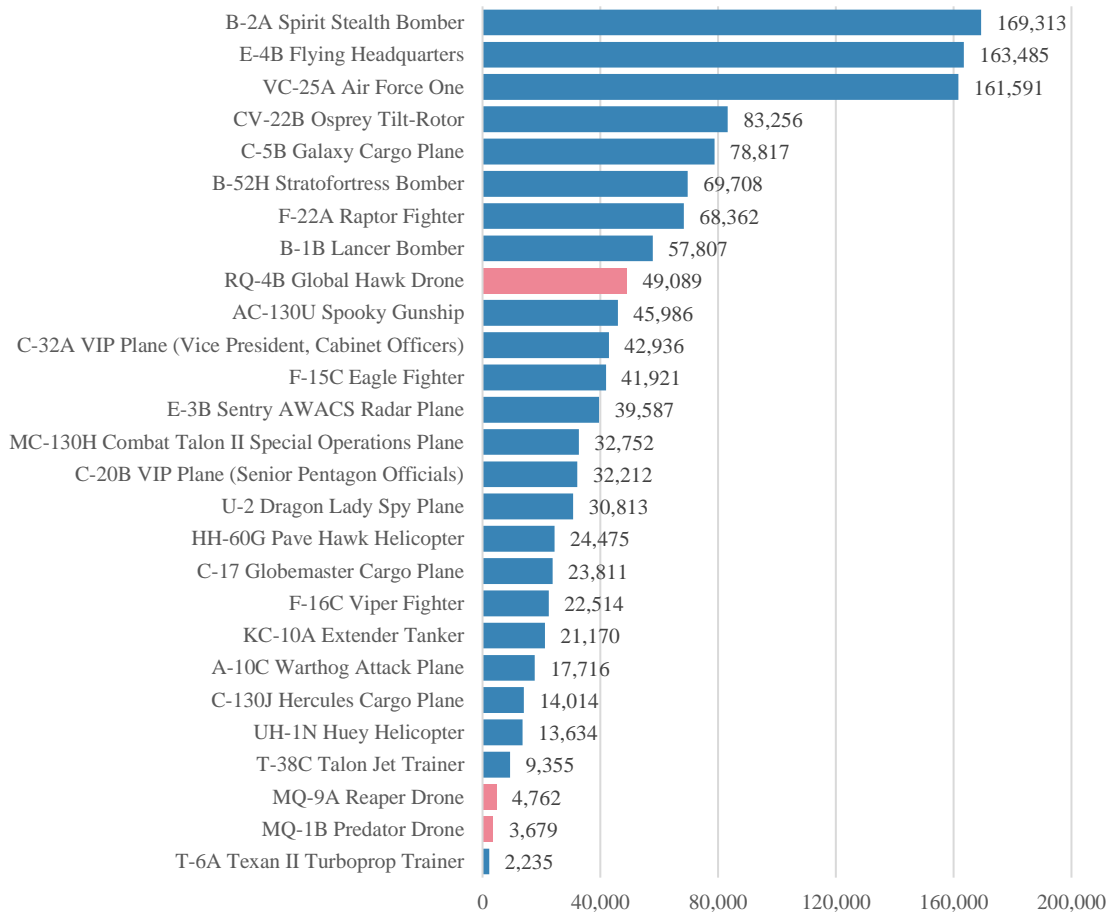


Figure 1.4: USAF aircraft cost per flight hour [27]

While the cost of the unmanned systems varies greatly from \$3,679 to \$49,089 per flight hour, they are the cheapest solution for a given mission. Indeed, the Global Hawk remains one of the most inexpensive solution in terms of very high altitude and long endurance intelligence gathering whereas the same observation can be drawn for the Predator and the Reaper drones in the domain of Medium Altitude Long Endurance (MALE) missions.

This trend is also observed in commercial applications, for instance in aerial photography and videography. Renting a fixed-wing aircraft or a helicopter typically

ranges from \$300 to \$1000 per hour [28], [29] with an additional fixed cost ranging from \$5,000 for simple shoots to \$20,000 for a mounted gimbal solution [30]. Taking again aerial imagery as an illustration, the operating cost is evaluated in terms of covered area. While imaging cost quickly soars for UAVs as the area to be covered increases, this latter solution still proves the cheapest and most precise for areas under 11 ha as shown on Figure 1.5.

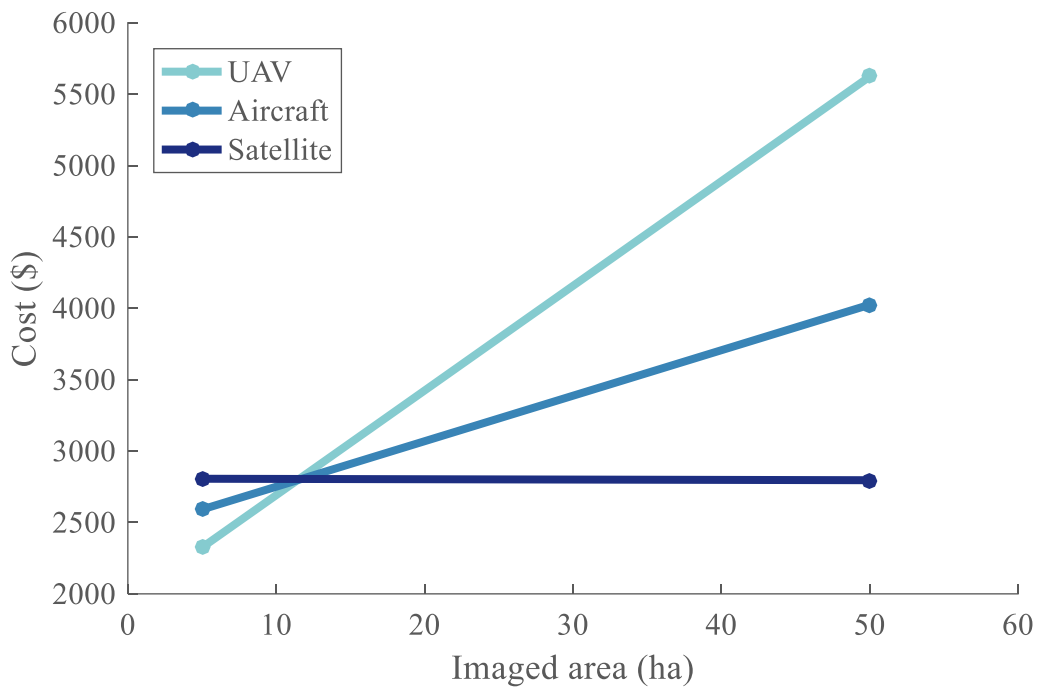


Figure 1.5: Imaging cost per platform [31]

This point can be further supported by looking at the difference of cost for given civilian applications as presented in Table 1.1.

Table 1.1: Manned vs. unmanned mission cost comparisons [32]

Mission	Manned System	Unmanned System
Sandhill Crane Population Survey	\$4,300 (government) \$35,000 (contractor)	\$2,600
Mesa County Landfill Inspection	\$10,000	\$300
Mesa County Gravel Pit Inspection	\$10,000	\$120

It can be noted that in the three missions examined, choosing unmanned systems results in reductions in the total cost from one to two orders of magnitude.

Finally, the advantages of robots over a human workforce mentioned in this section often give rise to manual labor replacement, incurring in turn additional cost savings. Indeed, human workforce can be replaced for any of the pre-stated benefits: the assigned task is too dangerous, strenuous, complex for a human being, or the exploitation cost of the robot is simply cheaper than the one of the human operator.

Summary: This section showed that robots generally offer many advantages over humans: they propose enhancements in rapidity, precision, and stability. This includes missions unsuited for human operators and they are also cheaper most of the time. These incentives partly explain the growth of the market of unmanned systems, studied in the next section.

1.2.2 A growing market

While the premises of Unmanned Aerial Vehicles (UAVs) were set at the beginning of the century, the market analysis proposed here concentrates on what are considered as

modern UAVs thanks to developments in robotics over the last twenty years. In the past decade, the unmanned vehicles market has experienced a tremendous growth in the military as well as in civilian applications (Figure 1.6). Their use has been constantly increasing in military operations since the terror attacks of September 11, 2001. They have now become an integral part of military operations, as demonstrated by the recent airstrikes campaigns in Syria and Afghanistan [1]. Comparably to many other cases, the expansion of unmanned systems for military use has also triggered an accelerated transfer and development of this technology into the civilian world. This democratization was fostered by the creation of public drone community forums such as DIY Drones in 2008 and the first large scale introduction of a public UAV model probably came with the release of the Parrot A.R. Drone model in 2010. Despite the current lack of in-place regulations for many countries and especially the United States, the growth of this market has been exponential for the last few years and is expected to continue as showed on Figure 1.6.

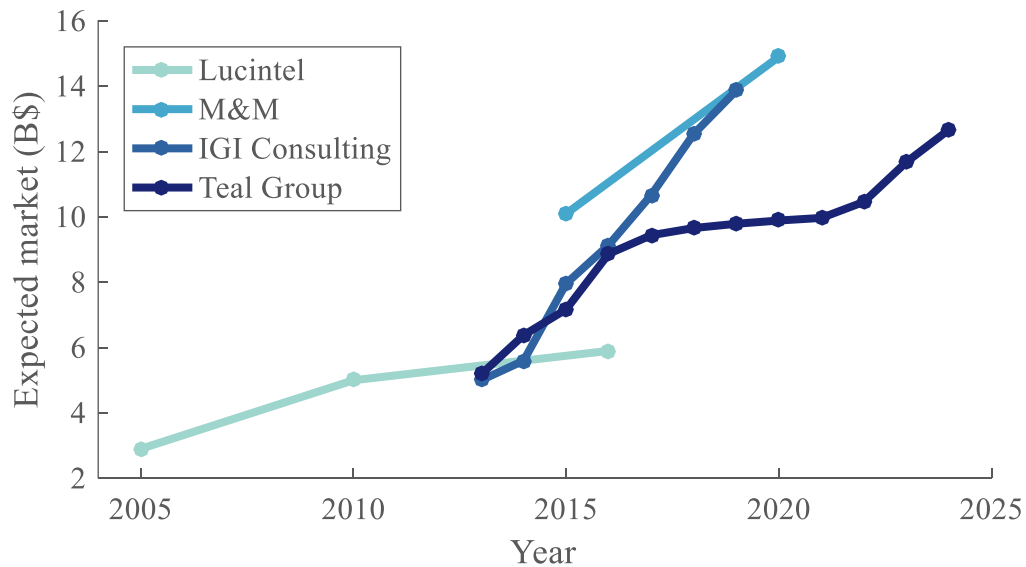


Figure 1.6: UAV global market forecast [33, 34, 35, 36]

Many other forecasts are available and all differ in their predictions, possibly by several orders of magnitude for the 2025 horizon [37]. Notwithstanding the absence of a clear consensus for the size of the market over the next seven to ten years, most consulting firms predict a strong growth. This lack of agreement between the forecasting parties stems from the difficulties of making clear predictions about the UAV market. Some of its segments are indistinct as some platforms are sold as general consumer products whereas they are also advertised as professional vectors to be used for commercial purposes. As a consequence, a distinction between the consumer and commercial drones market cannot be drawn and both are considered as the same segment. Another source of uncertainty in forecasting the drone market lies in the fact that very few countries have established legal regulations for the operation of unmanned systems in their national airspace. For the U.S., the Federal Aviation Administration (FAA) has been working tightly with companies and startups to help define the requirements of such airspaces. An example is the proposal by Amazon to have low altitude airspaces allocated for automated vehicles as well as no fly zones and free flight zones (Figure 1.7).

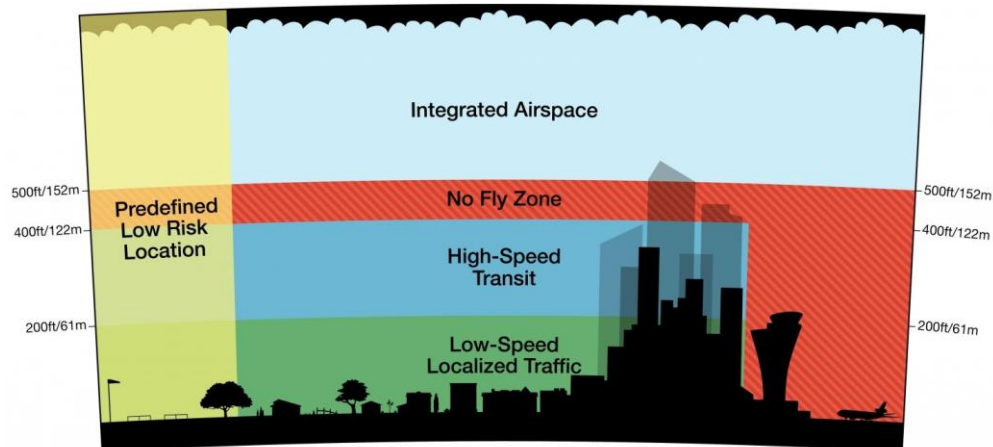


Figure 1.7: An example of drone airspace integration [38]

Since government policies control the use of the airspace and whether pilots can operate beyond visual line of sight, this makes the drone industry a regulated market for which forecasting is especially tough. In addition, the global drone industry did not wait for regulations to be set up and is betting on a technology that could take years to be legal and properly regulated [37]. This could possibly backfire on the market forecasts if the regulations turn out to be more stringent than what was initially expected.

This advance of the market translates in an augmentation of the number of vehicles in the National Airspace System (NAS): a proliferation of unmanned systems is expected (Figure 1.8).

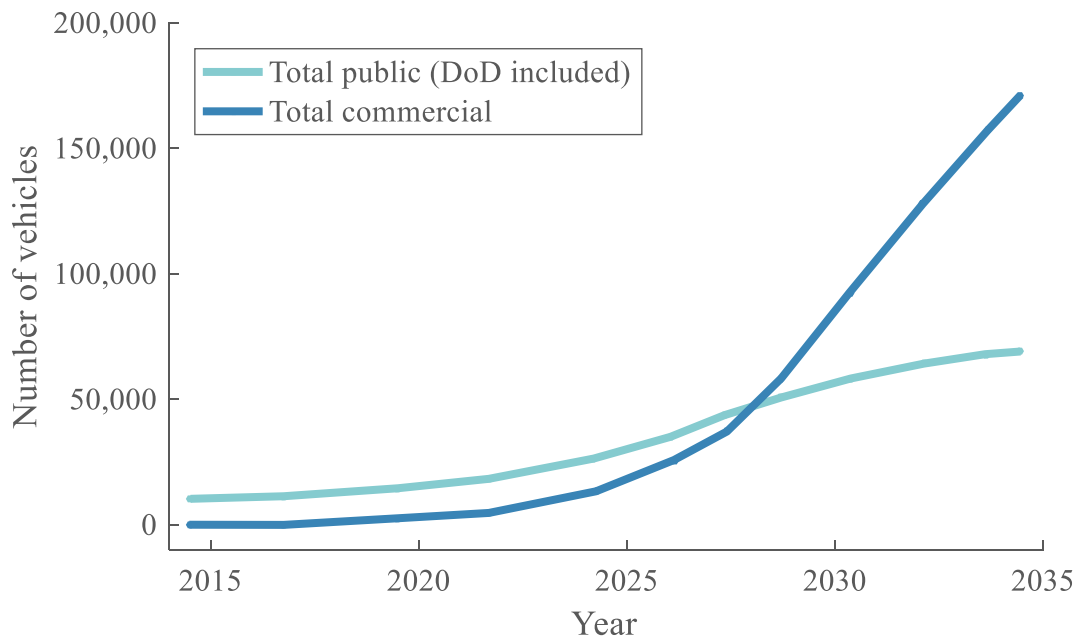


Figure 1.8: Total U.S. UAS Systems in the National Airspace System [33, 39]

The transfer from military to commercial use can also be noticed since the forecast for civilian applications grows at a faster rate after the 2025 horizon. This can be explained by the fact that UAVs can be used for many more purposes in the civilian world than in the military: the long term market will be dominated by the civil side. The military has indeed been acting as an early adopter for the technology, working on the maturation phase, demonstrating its utility, and encouraging the idea of its use in non-military applications. This population of vectors will most likely be dominated by MALE vehicles as it can be seen on Figure 1.9 while the part of High Altitude Long Endurance (HALE) will be limited. The Micro Unmanned Aerial Vehicle (MUAV) market will be the other field experiencing a growth, as opposed to Tactical Unmanned Aerial Vehicles (TUAVs).

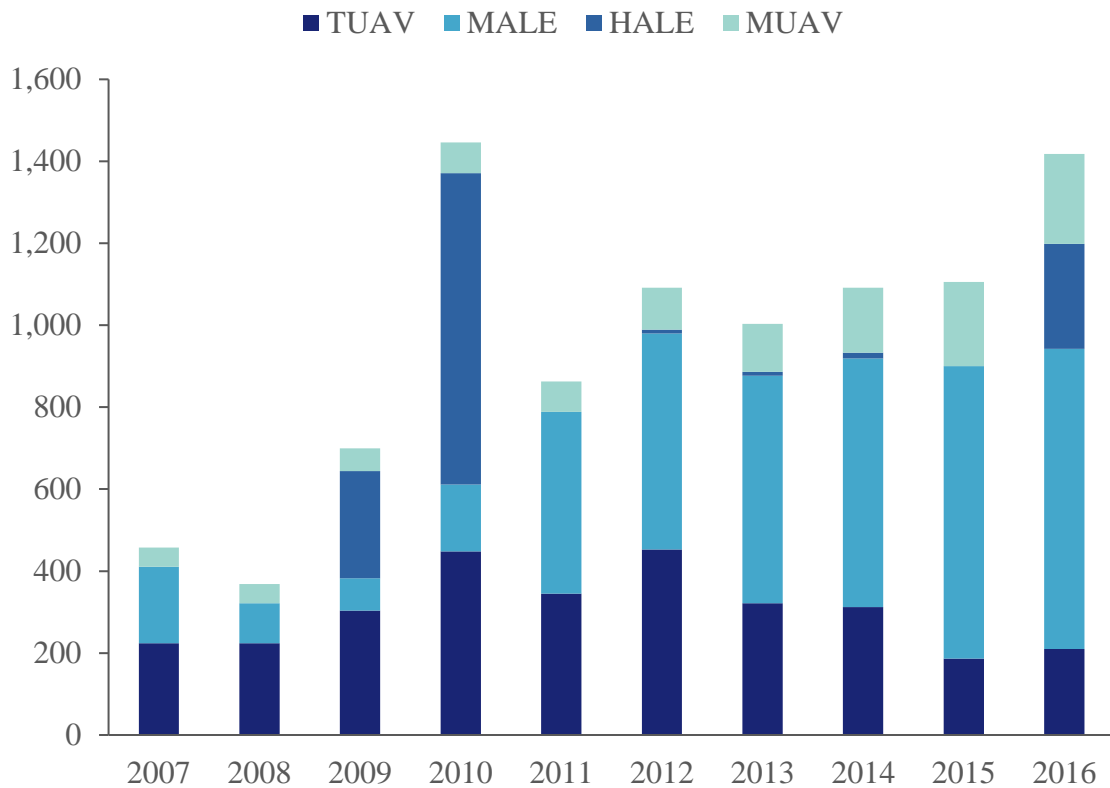


Figure 1.9: Expected military UAV revenues (Europe) [40]

In addition, the adoption of the technology by giant commercial companies such as Amazon, Google, or Facebook stimulates the emergence of the field and pulls smaller startup companies into the market emulation (Figure 1.10). With its Prime Air service, Amazon first started by advertising a revolutionary drone delivery program able to distribute packages within thirty minutes of an order placement. Google shortly followed by announcing its own drone-based delivery system named Wing, using a different architecture. Finally Facebook proposed, through a very high endurance design, a different utilization of drones aimed at providing remote areas with Internet.



(a) Amazon [41]



(b) Google [42]



(c) Facebook [43]

Figure 1.10: Drone programs from the giant tech companies

This growth is also facilitated by the maturation of the technology and the advancements in other related fields such as battery densities, electronics miniaturization, sensors development, as well as robotics intelligence. This facilitates the integration of each of these disciplines on the vehicle, resulting in an enhanced general performance. In addition, this drives the prices of drones to decrease, making them affordable to a wider public, enabling their adoption for a broader range of civil applications. To mention an expressive example, having easy-to-use agricultural drones equipped with cameras for less than \$1,000 represents for farmers a cheap way to tackle the increasing need of a data-driven agriculture: crops monitoring for better water use and pest management [44].

A good benchmark to look at for this price reduction is one of the first product released to the public market in 2010: the A.R. Parrot Drone. Over a period of two years up to now, its average price has decreased by 33% (Figure 1.11). Despite several factors affecting the manufacturing channel – such as the price of Lithium Polymer (LiPo) batteries, the average price decrease can be thought of as a PC pricing-decrease model [45] quite reminiscent of the rise of personal computing in the 1970s [46].

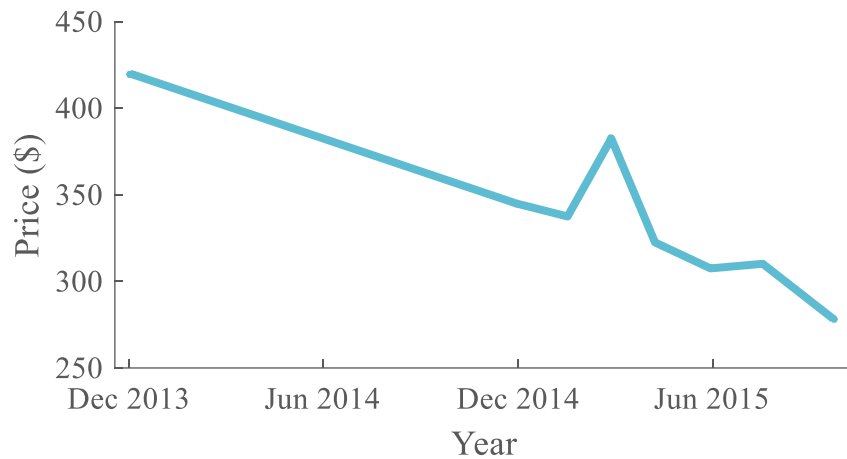


Figure 1.11: Parrot A.R. Drone 2.0 price evolution [47, 48]

The DJI Phantom drone and its different configurations constitute an additional benchmark as they are currently amongst the most popular platforms: from 2013 to 2015, an average reduction of 24% was observed on its price (Figure 1.12).

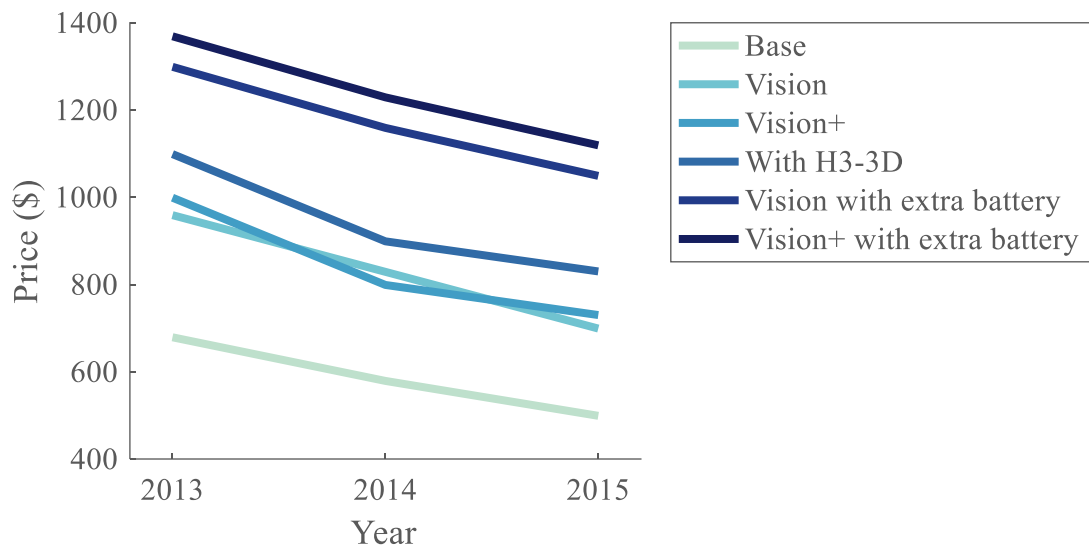


Figure 1.12: Price reduction for Phantom 2 variants [49, 50]

Furthermore, the DJI Phantom III, latest version of the drone, is expected to experience a price drop of 47% in the next 5 years [45].

Summary: In the face of the complications in obtaining accurate market forecasts and conflicting prediction models, it was established in this section that the unmanned aerial systems field is experiencing a tremendous growth. This applies for both the military sector as well as the public and commercial sectors. The recent involvement of major public companies further cultivates the progress in the sector. On top of that, the cost reduction in the manufacturing channel leads to lower prices, facilitating the adoption of the technology by a wider public. The fertile ground above-mentioned gives rise to a number of new unmanned systems configurations, focus of the next section.

1.2.3 A fleet getting more diverse

Owing to the introduction and rapid development of drone technology in the civil world, many applications are envisaged and require new designs. Indeed, new operation constraints encourage the need for increased vehicle capabilities and enhanced sensing technology. As the spectrum of drone operations is widening, classical architectures such as the quadcopter or the fixed-wing design become sub-optimal for original mission requirements and novel solutions are considered. A quick search reveals that more than 745 military drone models exist today [51] and this number is most certainly even higher for drones available to the general public. Focused mainly on unmanned aerial vehicles, this section will examine the reasons and the extent of such a diversity. Some of these observations also apply to ground robots and maritime unmanned systems.

As a consequence of the substantial amount of complex electronic equipment onboard, a main limitation of unmanned aerial systems is endurance and their design has become an exercise of energy conservation through optimization. This latter is hereby defined as the act, process, or methodology of making a design as fully perfect, functional, or effective as possible [52]. Each platform is thus optimized based on a specific application and for exclusive mission requirements. Hence, despite a relative robustness, a slight modification of the design or mission requirements will result in a suboptimal configuration. This observation will be illustrated by taking the example of a classic one-kilogram quadcopter designed for simple laser mapping purposes. If mission requirements suddenly include the ability to locate heat sources in the created map, a solution would be to add a thermal camera to the quadcopter. Such sensors are usually quite heavy and weigh

around one kilogram. However, the quadcopter was optimized for another payload and cannot accommodate an additional sensor like a bigger manned vehicle could have. Hence, in cases where a manned platform could have just been upgraded with an additional sensor, unmanned systems often require whole new designs to adapt to new mission requirements. This explains why many models and configurations of them exist.

Even for multirotor designs which might seem very similar at first glance, fundamental differences exist between a tricopter and a quadcopter, or a hexacopter and an octocopter. Configurations with many rotors are generally used for increased lifting capability and redundancy purposes, but require different control schemes. Focusing only on the quadcopter design, many designs and capabilities are available on the market. For example, as it can be seen on Figure 1.13 where fifteen representative models are listed, this variety is represented in terms of endurance and price, despite the fact that these platforms all have the same basic architecture.

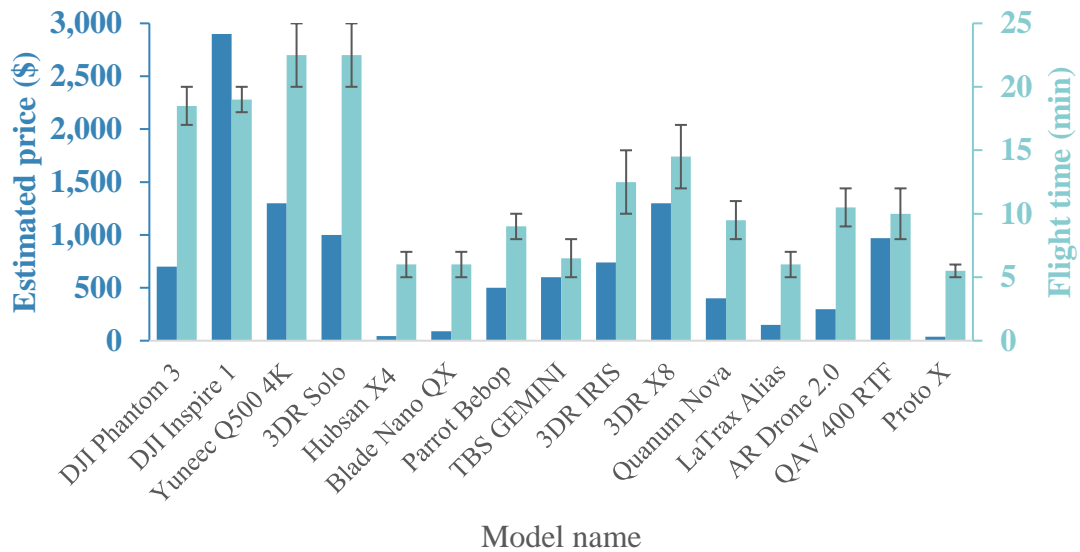


Figure 1.13: Civil micro quadcopter UAV market [53]

Besides the two typical designs and variations of the multirotor and the twin boomer, exotic architectures have started to emerge to respond to original mission requirements. Without having the pretention of listing all possibilities, a selection of unusual designs is provided here below. A first and simple new architecture consists in merging the capabilities of the two standard architectures: a hybrid design. Indeed in many missions, the ability to cover long distances is desired, as well as the hovering faculty. A notorious limitation of multirotor designs is their lack of endurance (Figure 1.13) since the propulsion system has to sustain the whole weight of the platform in the air: an energy-demanding task. Providing a multirotor with the ability to perform forward flight using an assisting wing alleviates the weight ratio to be carried by the propulsion system only. This hybrid design is already adopted by the delivery programs from Google and Amazon (Figure 1.10). The forward flight configuration is used to cover distance while the hovering capability is utilized for the precise delivery approach. Other unconventional architectures may include:

- **Non-planar multirotor:** for full control in the air in terms of both translation and orientation, orthogonal directions of action are required. A non-planar multirotor has rotors arranged in orthogonal planes, ensuring that the vehicle can hover and translate in any orientation, but also change its orientation at a given position in space. A simply impossible task for conventional multirotors.
- **Wall roller:** mostly designed for building inspection, this architecture can get very close to walls or ceilings and roll on them for detailed examination.

- **Balloon:** generally preferred for missions requiring extremely long endurance, balloons however lack speed.
- **Flapping wings:** also referred to as ornithopters, these architectures are especially coveted by the military for stealthy missions since flapping wings enable very small sizes for the vehicle.

Some of these alternatives are presented in Figure 1.14 here below.



(a) **Multirotor** [54]



(b) **Twin boom design** [55]



(c) **Helicopter design** [56]



(d) **Hybrid design** [57]



(e) **Tilting wing hybrid design** [58]



(f) **Non-planar hexarotor** [59]



(g) **Wall roller** [60]



(h) **Balloon** [61]



(i) **Ornithopter** [62]

Figure 1.14: Examples of classical and unconventional designs

Very recent research also includes vehicles able to evolve both in water and in the air, or on the ground and in the air [63], [64].

The emergence of such atypical designs is empowered by several key enablers, the first one of them being the existence of an active open-source community. Communities are essential as they facilitate the establishment of standards and norms for a given field, the UAV sector being one in a dire need for regulations. They also accelerate the spread of knowledge, critical in the case of sharing safety and privacy rules for the use of UAVs. An example of such a community is DIY Drones [65], indisputably the reference in terms of custom-made unmanned systems. Through forums and articles, communities put in relation novices with more experienced users who provide feedback and advice on the proposed designs. Such a public constitutes a fertile ground for the advent of new unconventional designs similar to those displayed on Figure 1.14. One of the major accomplishments of the DIY Drones community is to have created the first “universal autopilot” [65], virtually providing the ability to turn any Radio-Controlled (RC) model into a fully-autonomous UAV.



Figure 1.15: Interface of ArduPilot [65, 66]

Figure 1.15 shows that this universal autopilot allows for different architectures of vehicles: multirotor, contra-rotating multirotor, helicopter, fixed-wing, and even rover. The hardware itself is designed as a plug-and-play interface that can be just fixed on the vehicle and rapidly wired with the different actuators and motors. Its code being open-source, members have the possibility to modify it to account for custom architectures. In particular, many modify existing commercial platforms in terms of design and automation to suit their own mission requirements. This tool is now the reference universal autopilot and is used by a very large public: general consumers, RC hobbyists, researchers, and even commercial companies.

Although drastic changes in mission requirements often call for completely new designs, moderate changes can be addressed through few alterations of an initial baseline: an incentive for modularity. Indeed, it was established previously that the design of a UAV is quite fixed for a given mission. However, unmanned systems sometimes strike by their reduced size and even apparent simplicity, leading to wonder if slight design adjustments around central blocks could possibly address minor mission variations. Moreover, a broad spectrum of capabilities for a minimal cost is often a critical objective for recent projects. This question is addressed by the field called design for modularity: another key enabler for the diversity of the drone population. Using modularity, vehicles are reconfigured between sorties based on a library of interchangeable components that could comprise wings, tails, engines, and payloads [67]. For instance, a fixed-wing design is considered with a predetermined fuselage and empennage. The wing can then be optimized for speed, endurance, or transition flight in the same fashion that the payload can be customized for

imagery, communications, or surveillance purposes. Hence, three wings and three payload packages can exist and be clipped on a single vehicle core. The fuselage and the empennage can also possibly be declined in several configurations and all modules are quickly assembled to obtain the final vehicle (see Figure 1.16). Thanks to modularity, an initial design can be virtually duplicated in a spectrum of other configurations addressing different mission requirements.

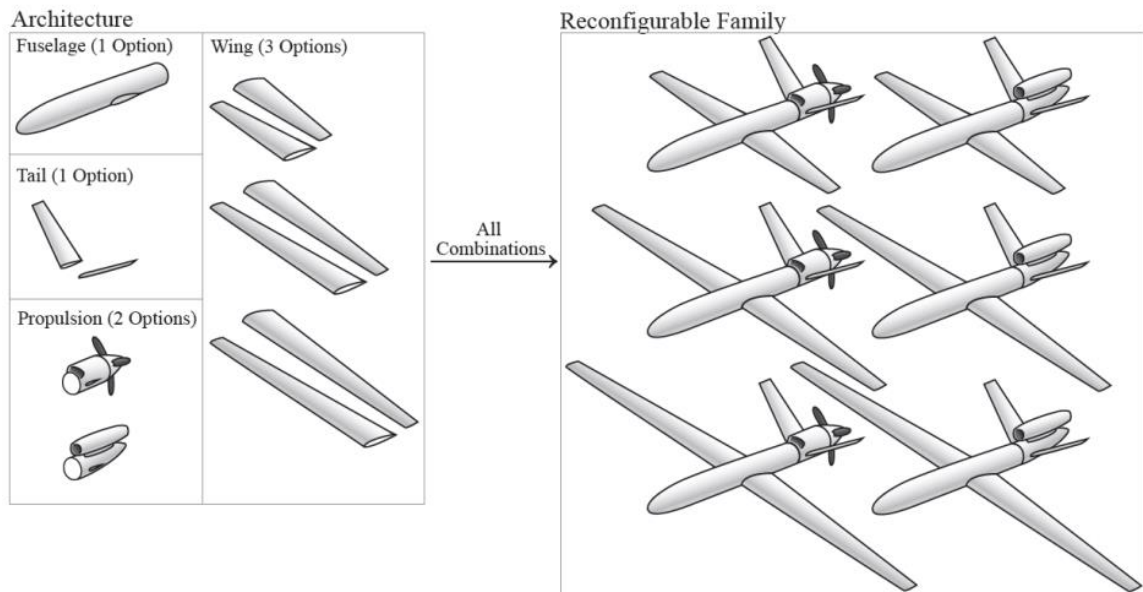
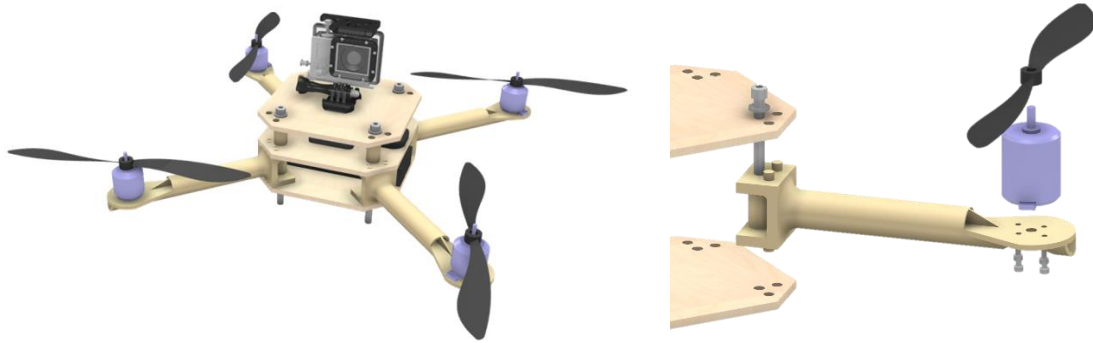


Figure 1.16: Notional family of reconfigurable aircraft [67]

An additional key enabler for the diversification of unmanned systems architectures is rapid prototyping [68], [69], [70], [71]. Mainly performed using 3D printing or additive layer manufacturing, it is a key enabler for modularity itself (Figure 1.17). These advanced manufacturing techniques enable the production of cheap reconfigurable parts when expensive molds were previously needed to do the same thing. This not only reduces the

cost of modularity in general but also the cost of unmanned systems in a broader perspective.



(a) Quadcopter with scalable 3D-printed parts

(b) Detail of the arm connection

Figure 1.17: Use of 3D printing on a quadcopter design [69]

It also permits the manufacturing of complex structures that are often the result of design optimization algorithms. For instance, complicated structural meshes such as the honeycomb mesh can be used to reduce the weight of some components like the wing (see Figure 1.18) while maintaining its structural strength.

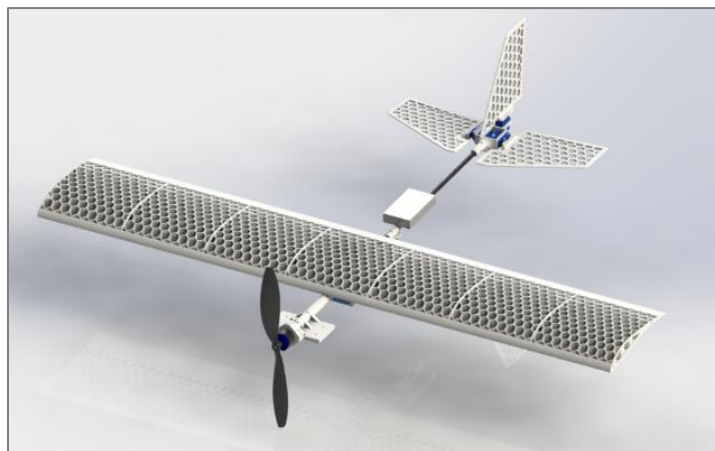


Figure 1.18: Use of 3D printing on a fixed-wing design [70]

Building on these observations, a first conjecture can be drawn:

Conjecture 1

There is a potential to take advantage of the upward diversity
of the existing drone fleet.

Summary: This section established on one hand that unmanned systems are subject to stringent design constraints, notably on endurance and payload weight. This motivates the need for a variety of models in order to be able to carry out the myriad of missions anticipated by the growing market. On the other hand, it was showed that thanks to their reduced cost, size, and sometimes complexity, unmanned systems designs are predisposed to modularity and customization for evolving mission needs. Each design is then virtually duplicable in an infinity of novel models, partaking in the population diversity. These observations, coupled with the power of an open-source community as well as rapid prototyping, were identified as key enablers for the proliferation of architectures for civil applications and military purposes. In the next section, the limitations that could possibly hinder the promising potential of unmanned systems are examined in detail.

1.2.4 The limitations

As a result of their particularity, unmanned systems are exposed to limitations and constraints that manned systems might not encounter. The most widespread ones are their lack of endurance, especially for micro-sized UAVs, and their deficiency in cognitive behavior. Other drawbacks of unmanned systems, which motivate a progression towards the research objective of this work, are also considered in this section.

On account of their often reduced size, unmanned systems are subject to limitations in terms of the payload they can carry and this especially affects their batteries. As a result, unmanned systems often exhibit a lack of endurance for the missions they are assigned to. This applies particularly to micro-sized platforms and Micro Air Vehicles (MAVs) whose size is of the order of thirty centimeters. Observing the example of quadcopters, these are quite often used for mapping purposes but barely exhibit an autonomy of more than fifteen minutes (Figure 1.13). This endurance remains insufficient to takeoff, perform the calibration tasks, reach the mission area, build the map, exit the building, and finally return to base. By looking at the history of the state of the art lithium-ion batteries in terms of maximum capacity, it is possible by a simple rule of thumb to compute what would have been the equivalent maximum battery life of a DJI Phantom over the years (Figure 1.19).

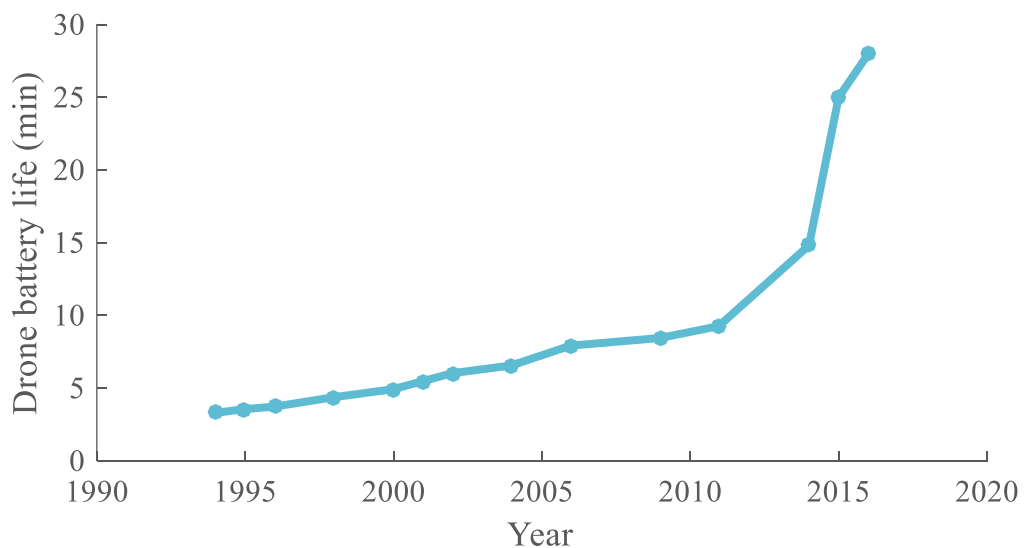


Figure 1.19: Equivalent DJI Phantom battery life over time [72, 73, 74]

Although this battery life has experienced subsequent improvements over the past few years due to design optimizations and advancements in battery technology, it is still very limited when compared to what bigger platforms are able to accomplish. Even microdrones with fixed-wing configurations, which are supposed to last longer in the air thanks to additional lift, still exhibit flight times under the hour [24]. Non-flying platforms are also affected in the same fashion while they do not have such high requirements in energy draw as their flying counterparts. This is a clear limitation for the missions that unmanned systems are set to carry out.

The limited endurance mentioned hereinabove remains firmly coupled with the amount of computational power and cognitive capabilities that it is possible to embark on a robot. Indeed, the payload defines the power consumption required in operation not only by its own current draw but also sometimes by its weight for aerial vehicles: hence doubly affecting battery sizing. The design is thus an iterative loop since additional payload requires additional power to stay in the air, which calls for more batteries, inducing an increase in weight which in turn implies a more important need for power, and the loop continues. For this lack of embarked computational power and hence cognitive capacities, it is often said that robots are very good for things that humans consider as complex or hard, but demonstrate quite poor performance for tasks deemed as standard. For instance, it is very easy for an unmanned system to compute the exact distance to an object in its environment, a task for which a human agent could only give a rough approximation just by looking at it. In like manner, a robot can effortlessly perform thousands of mathematical operations in no time, or lift a thousand pounds of the ground: operations for which humans

would experience great difficulty. However, unmanned systems often look clumsy and uncertain when entering doors or windows whereas humans seamlessly walk through them, instantly incorporating the information from all their senses. Robots similarly require consequent computation time and complex algorithms to identify people or objects, a task easily accomplished even by young children. In addition, the transition between the different stages of a mission frequently turns out to be slow due to recalibration processes. This lack of cognitive behavior on individual agents stems from a deficit of computational power and full situation awareness which could be addressed by having other units contribute to the tasks or sharing complicated processes across several robots.

The latter also impedes the autonomy of unmanned systems, making fully autonomous missions almost impossible without a human supervision, causing the task to be only semi-autonomous. Indeed, taking the example of military operations, local operators manage the takeoff and landing phases directly at the airfield before putting the asset on a holding flight pattern before remote operators based in the U.S. take over and carry out the mission manually: the fully automated part of the mission is very limited in terms of complexity. As a result, unmanned systems are inclined to stringent communication link requirements so as to maintain constant supervision of the whereabouts of the asset. This required communication link can be particularly hard to set up, especially due to the very nature of the settings where unmanned systems operate: environments with difficult or dangerous access for human beings.

Unmanned systems currently lack a dedicated and protected radio-frequency spectrum required to ensure secure and continuous communications for their operations.

This worsens the probability that an unmanned aircraft will be vulnerable to unintentional or intentional interference [75]. The RQ-170 Sentinel stealth drone incident of 2011 showed that, because of this communication link requirement, even military platforms can supposedly be hacked [76]: an additional limitation to the use of unmanned platforms and their communication protocols. Military applications have their own protocols such as the Joint Tactical Radio System (JTRS) and even Long-Term Evolution (LTE) networks as they provide a superior mobile bandwidth and low latency, ideal for datalinks transmitting image and video information. LTE technology also enables the military to take advantage of a wide network of vendors and it will most probably be the future for Unmanned Aircraft Systems (UAS) communications [77]. However, most commercial and publicly available unmanned systems presently use radio frequencies channels around 2.4 GHz, a bandwidth compatible with Wi-Fi and thus with most mobile electronic devices. While this facilitates the integration of UAS with existing technologies such as smartphones and laptops, the range of such a solution is limited to a couple kilometers in the best case scenario and is at risk for interference and hacking.



Figure 1.20: Network settings of the Parrot Bebop drone [78]

As Figure 1.20 shows, a non-military UAS network might have to be set up on congested frequencies, especially in crowded areas. Using unprotected radio spectrum and wireless technology for the use of unmanned aircraft systems constitutes a major security and safety vulnerability. Indeed, disconnection of the communication link amputates the UAS of its only means of control, as against manned systems in which a pilot is directly in physical control of the aircraft.

Lost link is also a possibility when the command and control communication fails between the UAV and its ground station due to environmental or technological complications. Even though unmanned systems generally have pre-programmed procedures to hover and recover the signal or safely return to base, no standardization has been proposed by the FAA and air traffic controllers deal with the problem on a case-by-case basis [75]. This is another hurdle in the quest for a standardized communication protocol across all types of UAS.

On top of these previous, quite direct, limitations, unmanned systems are also subject to more discreet limitations due to the uniqueness of each robot. Indeed, the accumulation of capabilities on a single vector in order to increase its flexibility might end up either oversizing it for simpler missions, or making it simply too expensive to use. A good example of this assumption is the Fukushima nuclear incident of 2011, when many robots were deemed too expensive to be put to work at the power plant. For many advocates of automation, a perfect use case of robots is in case of a nuclear disaster for which humans cannot be sent on the scene to look for survivors or evaluate and repair the damage. Yet, after March 2011, none of the two robots designed for radiation possessed by Japan, a

country specialized in robotics breakthroughs, was sent on site [79]. Several plant operators decided that robots were too expensive. The research community, who usually sends some of their projects on disaster relief sites [80], also deemed that its platforms were too costly of an asset to lose to radiation as it had already happened at the Chernobyl incident of 1986 [81].

Furthermore, the U.S. Department of Defense has issued a roadmap for the integration of unmanned systems, underlying the factors that will influence their development in the near future [82]. The roadmap points out the expeditious integration of unmanned technologies in battlefield capabilities due to urgent operational needs after the 9/11 terror attacks. This results now in a critical need for interoperability and modularity amongst existing platforms. Indeed, sensors, computers, and algorithms are evolving rapidly, sometimes outpacing implementation capabilities. It is not rare that a new onboard computation chip is released while its previous version has just been integrated with success on existing vehicles. The Department of Defense proposes to reach this goal of modularity and interoperability by addressing intra-platform challenges as well as inter-platform challenges. Intra-platform modularity refers to the ability to easily transition upgraded capability or hardware onto fielded systems in a plug-and-play fashion. While this approach calls for an assimilation of modular design methods, it is important to note that the modularity feature can also be achieved by putting to use the existing platforms in a collaborative fashion. For example, consider a system of two robots 1 and 2, both equipped with a computing unit A (see Figure 1.21). This chip has enough power to run one algorithm only, either mapping or surveillance. A new chip B is released with increased

computing capabilities when compared to chip A. In order to perform mapping and surveillance, a designer could choose to create a new platform 3 running both missions thanks to the upgraded chip. This new vehicle would have to interface with the new computing unit. However, the same capability of simultaneous mapping and surveillance can be achieved by making vehicles 1 and 2 collaborate properly.

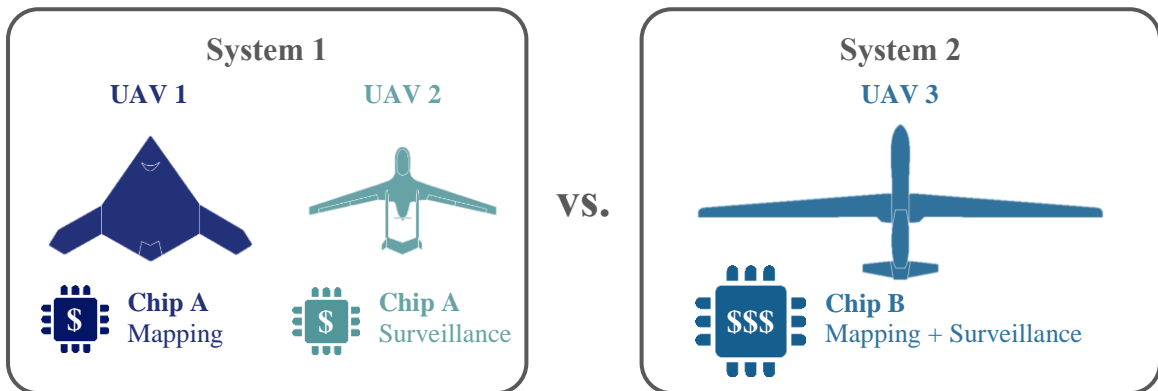


Figure 1.21: Taking advantage of fleet diversity [83, 84]

This integration of existing disparate assets is cost-effective since both vehicles are already developed, a key benefit for the Department of Defense.

Besides, UAVs with advanced purposes tend to require a lot of manpower to fulfill their mission. Most of them require at least a pilot and a sensor operator as direct operators, not to mention the necessary personnel for deployment and communications. For example, a single Predator UAV involves one pilot and two sensor operators, and it is estimated that it takes around 82 personnel in total to run a mission [85]. The aim of the Air Force is to have a ratio of around 10 crews for every drone combat air patrol [86], and even though

the actual ratio is between 8.1 and 8.5, it is still quite high compared to the requirements of manned missions. The total personnel for these combat air patrols is estimated to 186 and presented on Figure 1.22.

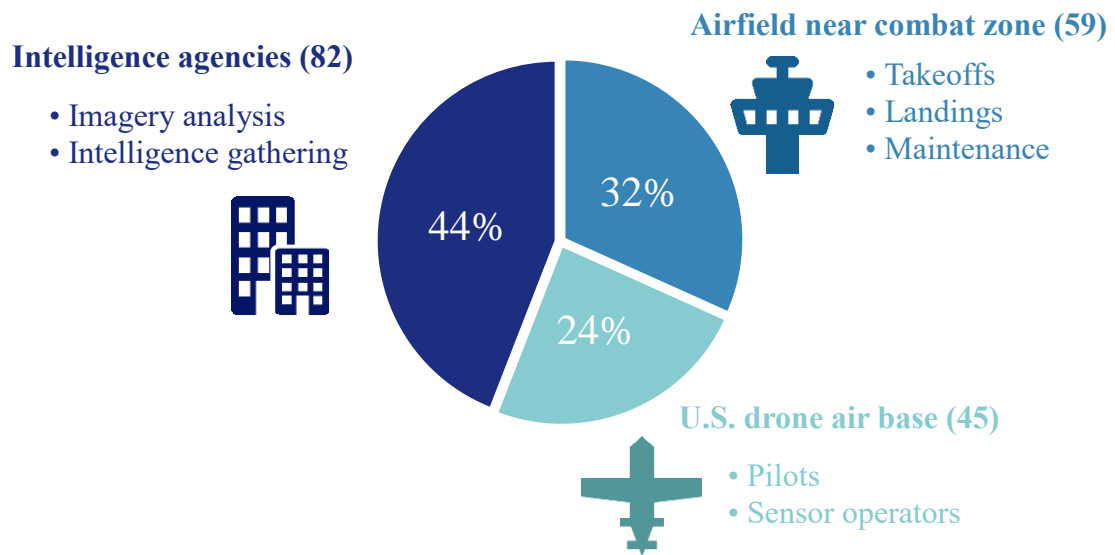


Figure 1.22: Total personnel for drone combat air patrols [86, 87, 88, 89, 90]

This number is justified by the additional security measures required to operate unmanned systems as opposed to manned vehicles and seems quite startling in view of the automation purpose of UAS. With the contribution of enhanced cognitive capabilities as well as communication protocol standards, many tasks could be automated and streamlined in order to reduce this important inertia to perform a drone mission. In contrast, it is important to notice that most civil and commercial UAS require one operator most of the time to carry out a mission.

Finally, the public opinion regarding drones suffers from several misconceptions and concerns which are slowing down the democratization of the industry for civil applications. Indeed, such platforms have widely been used, and advertised, for national defense so far which makes it harder to imagine their non-military benefits. Until a few years ago, there had been little acknowledgment of the use of unmanned systems for agriculture, mapping, search and rescue, and humanitarian or disaster response. The fact that it took more than a year for the FAA to establish a first set of rules, undoubtedly insufficient, for the regulated use of unmanned air vehicles clearly, reveals that some unease remains regarding their use in the civil world.

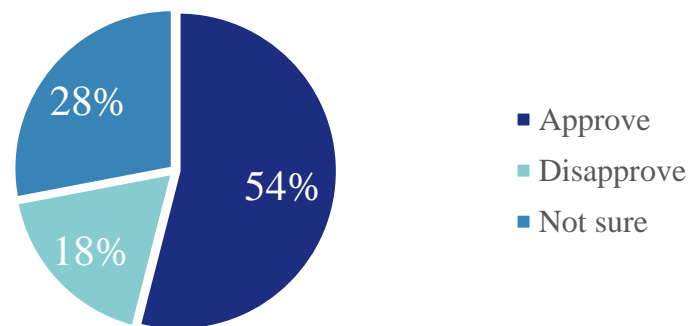


Figure 1.23: Approval of drone use to kill high-level terrorism suspects overseas [91]

General misconceptions highlighted by [92] are that UAS are dangerous to manned aircraft and to people on the ground as they might collide and crash in populated areas. They are also commonly considered as privacy threats and some residents sometimes shoot drones trespassing over their property [93]. In fact, even the military drone program is widely criticized for killing remotely and indiscriminately and does not show a clear

support from the American population (see Figure 1.23). These misconceptions leave the UAS industry with a tricky case to make to show the positive impacts of drones.

Summary: This section highlighted that unmanned systems are subject to limitations especially in terms of endurance and computational power. The induced lack of cognitive behavior calls for an essential supervision. This latter is provided through complex communication links which can be hard to put up in the operational environments of unmanned systems. In addition to exposing the vehicles to hacking hazards, it sometimes also requires additional manpower to be operated, incurring additional costs. Finally, their integration in real-life operations was done quite hastily, leaving room for some improvement in interoperability. The observations made throughout this section also show that in some cases, using the capabilities of several unmanned systems could have liberated the overall mission from several limitations: the broad spectrum of unmanned systems architectures is not put to use. Finally, the public perception of unmanned systems is negatively connoted which hinders the expansion of the industry. The next section examines how some of these limitations can be tackled by making several individual robots collaborate in a group.

1.3 The growth of multi-robotics

As it was hinted in the previous section, some of the limitations of individual unmanned systems can be lifted by making several of them collaborate. The complexity of some environments or missions might require a combination of robotic capabilities which are too expensive to design on a single platform [81]. Cooperative teams of robots are studied by the field of multi-robotics where many individual agents work together in a

coordinated fashion to accomplish a unique goal. Such solutions enable the entire system to respond robustly, reliably, and flexibly to unexpected changes in the environment or in the robot group itself. In particular, swarm robotics is a particular approach to collective robotics and is inspired from the self-organized behavior exhibited in nature by social animals such as fishes, birds, ants or bees. The main idea of swarm robotics is to obtain a complex synergistic behavior through simple behavior rules and local interactions.



Figure 1.24: Example of robotic swarm [94]

Swarm robotics use simple robots compared to the complexity of the mission to perform. The increase in capability over single robots comes from the collaboration aspect. Since several robots are used, this gives robustness, flexibility, and scalability to the swarm. This approach of collective robotics also represents an interesting way to capitalize on the heterogeneity of the current fleet of robots described in the previous section. For its future projects, the DoD stresses in its UAV integration roadmap the importance of several factors such as interoperability, modularity, resilience, autonomy, and cognitive behavior [82]. The present distribution of operations is creating a need for robotic systems to become

increasingly interoperable. These characteristics match with the capabilities offered by multi-robotics. Robot groups can be used in real-world applications for exploration, surveillance, search and rescue, humanitarian demining, intrusion tracking, cleaning, inspection, and transportation of large objects [95]. However multi-robotics is a complex field and several of its limitations are exposed at the end of the present section.

1.3.1 A field inspired by nature

The observation of animal behavior has been used as a source of inspiration for the development of individual robots and can also be used to gain insight into the creation of cooperative groups of robots [81]. Hence, it is no surprise that the current research underway in multi-robot systems can be classified quite similarly to the way animal societies or colonies are. Indeed, a first consequent body of work concentrates on very large numbers of robots that have limited individual capabilities but can generate complex behaviors through cooperation: swarming systems. A second approach which can be referred to as “intentional” cooperative robotics – or more generally cooperative robotics [81], emphasizes higher-level intentional cooperation between robots that have a higher level of intelligence. The first category can prove particularly suited for very repetitive tasks over large areas while the second one may be more adequate for applications with several distinct tasks. Both these approaches can possibly overlap in their missions, or even assumptions, and they are described in the following paragraphs.

1.3.1.1 Swarm robotics

Particular subcategory of multi-robotics, swarm robotics has been gaining special interest in the past decade. It takes its inspiration from societies of insects in which the swarm can perform tasks that are beyond the capabilities of the individuals. Such collective

behaviors are observed in the dance of honey-bees, the nest-building of wasps, the construction of mounds by termites, or in ants following trails. Figure 1.25 presents other examples of such comportments. These latter were considered as mysterious for a long time in the biology field and it was showed in the past few decades that individuals do not require sophisticated knowledge to exhibit complex behaviors as a swarm [96]. Indeed, there is no leader guiding the colony to accomplish an established goal and each agent does not know the overall status of the swarm. The global knowledge is distributed amongst all individuals which cannot accomplish the task without the rest of the group.



a) A ball of mackerel fish defends against sea predators [97]



b) A bird swarm [98]

Figure 1.25: Swarm behaviors in nature

The agents are able to exchange information locally and these interactions modify their behavior based on the previous changes made by their mates in the environment. The self-organizing comportment emerges from the propagation of information throughout the swarm. This former has external communication between its agents and their environment, as well as internal communication between the agents themselves.

A couple detailed examples of swarm behaviors can be examined. For instance, ant colonies can locate food sources the nearest to their nest without any single ant having this precise information. Thanks to chemicals called pheromones laid down on the path to the food source, ants can identify the shortest path. Indeed, ants returning first to the nest most likely took the shortest path and other ants hence follow the same path, reinforcing the trail of pheromone which encourages the colony to take this track [99]. As for bees, they have scouts exploring areas around the nest to find decent locations for a new nest. Each scout returns to the cluster and share their findings thanks to a dance, encouraging others to explore the location. Once more individuals are convinced about the location, a favorite location starts to emerge and the whole swarm finally flies to it. This swarm behavior ensures the safety of the cluster and is quite efficient in finding the most suitable nest site [100].

The fields of physics and chemistry present theories explaining how complex collective behavior can emerge from interactions of agents behaving simply [96]. Swarm robotics tries to mimic these behaviors to make simple robots accomplish complex missions that cannot be carried out by a sole agent. In so doing, it has identified the desirable properties of swarms of social insects to apply to swarm robotics. The first one is robustness, the ability for the swarm to still function correctly if some of the agents fail or if the environment experiences disturbances. Since robots can sometimes be expendable assets as it was established in the first section, it may happen that a few agents are lost during a mission. However, as opposed to single unit missions where the mission would stop, a swarm should continue to operate with a reduced number of agents. A second

quality desirable for swarm robotics is flexibility: the swarm should be able to adapt to different tasks and reallocate the roles of its agents based on the needs of the moment. This is especially true if some agents of the swarm have diverse capabilities, a different layout of the subtasks for the individuals of the swarm can result in a different synergistic goal being accomplished. The last property observed in animal swarms and that is advantageous for swarm robotics is scalability: the capability for the group to perform a mission independently of the size of the swarm, from a few individuals to thousands of them [96]. This interesting aspect of swarm robotics enables to seamlessly add one or several robots to the swarm and having them immediately contribute to the overall mission without overwhelming reprogramming tasks. This can be done to improve the performance of the swarm on a given task for instance.

Given these properties, a first definition of swarm robotics can be given [96]:

Swarm robotics is the study of how a large number of relatively simple physically embodied agents can be designed such that a desired collective behavior emerges from the local interactions among agents and between the agents and the environment.

Based on [95], a set of conditions complements this characterization to establish whether a multi-robot system constitutes a swarm or not:

- i. The robots of the swarm must be autonomous robots, able to sense and actuate in a real environment.
- ii. The number of robots in the swarm must be large or at least the control rules allow it.
- iii. Robots must be homogeneous. There can exist different types of robots in the swarm, but these groups must not be too many.
- iv. The robots must be incapable or inefficient with respect to the main task they have to solve, this is, they need to collaborate in order to succeed or improve the performance.
- v. Robots have only local communication and sensing capabilities. It ensures the coordination is distributed, so scalability becomes one of the properties of the system.

These properties can be compared to taxonomies found in the literature of multi-robotic systems (see Table 1.2).

Table 1.2: Swarm robotics taxonomy axes [101]

Axis	Description
Collective size	Number of robots in the collective.
Communication range	Maximum communication range.
Communication topology	Of the robots in the communication range, those which can be communicated with.
Communication bandwidth	How much information the robots can send to each other.
Collective reconfigurability	The rate at which the organization of the collective can be modified.
Process ability	The computational model used by the robots.
Collective composition	Are the robots homogeneous or heterogeneous?

The discrimination between swarm robotics and other multi-robot approaches can be tricky to establish [95] especially when heterogeneous swarms are considered since they might violate assumption (iii). This explains the divide in the literature about unique definitions, characteristics, and taxonomies of swarm robotics as proposed by [102], [103], [104], or again [105].

Finally, one may notice that robotics is not the only field trying to apply such observations on biological swarms to science. Particle Swarm Optimization (PSO) for instance, tries to apply swarm behavior rules to the field of optimization. A population is randomly scattered in the design space and at each iteration of the algorithm, each agent

moves according to simple rules (see Figure 1.26) until the population converges on the optimum of a given cost function over the design space.

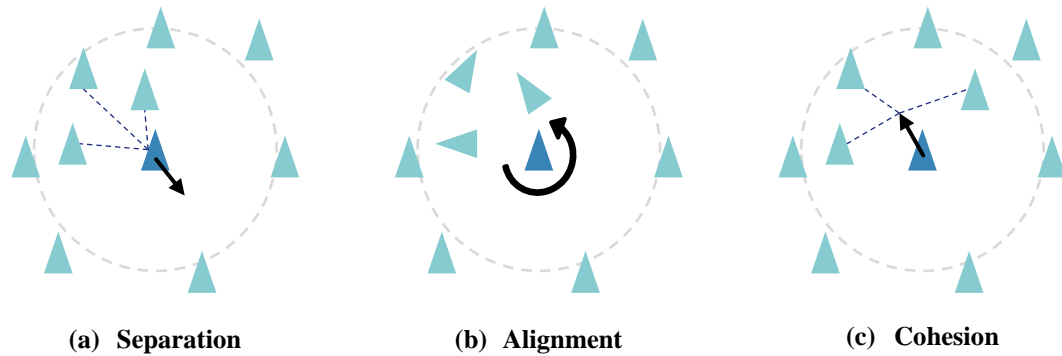


Figure 1.26: Swarm behavior rules in Particle Swarm Optimization [106]

These elementary rules are the following:

- **Separation:** each agent tries to maintain a minimum distance with its surrounding neighbors.
- **Alignment:** each agent steers towards the average heading of its neighbors.
- **Cohesion:** each agent tries to go towards the average position of its neighbors.

The resulting complex behavior from these simple rules is a performant optimization method able to efficiently find global optimums where many other methods fail.

1.3.1.2 Cooperative robotics

When time or efficiency constraints are placed on the mission, or when several distinct tasks have to be performed, a more directed type of cooperation might be required between the robots [81]. These missions generally involve smaller groups of agents when

compared to swarm robotics, and are more inclined to encourage heterogeneity. Instead of using a functional decomposition of the robot as with the swarming approach, most of the research on cooperative “intentional” robotics uses a more physical decomposition. The approach is then separated into world, sensors, planning, and action modules, also known as the sense-model-plan-act Artificial Intelligence (AI) paradigm [81]. The main focus of this type of approach is to determine through analysis the proper action and coordination scheme among the members of the group, in order for them to complete their mission. The research community is mainly split into two bodies: a first one focusing specifically on robotic implementation, and a second one applying Distributed Artificial Intelligence (DAI) to more generic types of agents.

In the first body of research, the sense-model-plan-act architecture is implemented with different layers of control. [107] hence proposes three layers: the planner level managing tasks coordination and allocation, the control level ensuring proper task execution by each robot, and the functional level for controlled reactivity. The method is demonstrated on a box-pushing task with a group of two robots. Another organization of layers is proposed by [108] using Petri Nets: a task planner, a task allocator, a motion planner, and an execution monitor at the lowest level.

Other approaches utilize subscriptions and requests mechanisms to coordinate tasks between the different robots of heterogeneous groups. For instance, [109] implements a negotiation framework which robots use to recruit help when needed. A sign-board method is used in [110], in addition to mutual exclusion algorithms.

By being centralized, these approaches lack fault tolerance and it is probable that a failure in one of the robots or in the communication protocol would result in a failure of the whole system [81]. Moreover, it makes it hard for multi-robot systems to deliver real-time performance in dynamic environments due to the difficulties and limitations of an implementation on physical robots.

The second body of research is not as focused on robotics implementation but proposes solutions to cooperation and conflict resolution for generic agents. These approaches qualify as DAI and generally use negotiation-based mechanisms to perform the task allocation between the agents. Such methods mentioned by [81] have agents broadcasting requests for bids to perform the different subtasks of the mission. Agents which are available, suitable, and willing to perform the tasks then respond to these requests and the initiator selects the most suitable agent from the pool. DAI solutions have shown successful implementations in a number of fields such as vehicle fleet monitoring and air traffic control [81] but they have rarely been applied to groups of robots. Indeed, such approaches usually rely on perfect world modeling assumptions with perfect sensing and action capabilities, which does not correspond to the situation in which robots and their noisy sensors and actuators evolve.

Summary: In this section, the essence of multi-robotics was captured from example behaviors found in nature such as with swarms of ants or bees, bird flocking, or fish shoaling. It relies on the fact that an elaborated collective behavior can emerge from a group, surpassing the capabilities of its individual agents, using simple local interactions

and comportment rules. When applying these observations to the field of robotics, three main characteristics are desirable: robustness, flexibility, and scalability. Example definitions, assumptions, and characteristics of cooperative and swarm robotics were established, despite a lack of consensus in the literature. The current work distances itself from some of these assumptions in order to simplify the approach and focus on the design methodology, hence evolving towards a more classical multi-robot problem, facilitating heterogeneity. Supported by the different opposing views on its definition, a swarm is here referred to as a group, possibly heterogeneous, of robots interacting to solve a complex task for which the constituents are individually inapt. This definition will constitute the main focus of this research and the words group and swarm will be used indifferently based on this definition. The enhancements in mission competence that it is possible to obtain from multi-robotics are then studied in the next section with implementation examples.

1.3.2 An increase in capability

The main advantage of multi-robotics lies in the very definition of it: obtaining a complex behavior from the collaboration of simpler individual robots. In other words, it presents the potential to bridge a gap in mission capabilities while making a minimum investment on the development of an individual platform. It is then possible to unlock novel abilities using a fleet of robots that is already at disposition. In addition to the three major defining advantages mentioned in the previous section (i.e. robustness, flexibility, and scalability), the different enhancements in robotics capabilities proposed by multi-robotics can be separated in different categories [111].

Task enablement: in some cases, using a group of robots unlocks new possibilities for missions that cannot be carried out by single robots. The most common example of this enhancement is obstacle clearance using group reconfiguration [112]. Figure 1.27 shows an example of how a group of robots can reconfigure itself in order to bridge a gap between rocks in the environment.



Figure 1.27: Holes avoidance through group reconfiguration [112]

The robots of the group self-assemble using physical interconnections to constitute a bigger entity capable of tackling the difficulties of the environment such as gaps, narrow passages, or obstacles. Not only the sensing done by each of the robots helps the group in understanding the nature and properties of the obstacle, but the robot itself is physically used by the group to pass the obstacle.



Figure 1.28: Chain formation for a narrow passage [112]

In Figure 1.28, the chain of robots navigates through a narrow and deep passage thanks to physical interconnections. Without this group behavior, each individual agent would fall in the passage and break while the whole group is able to exert more traction force and carefully negotiate its way.

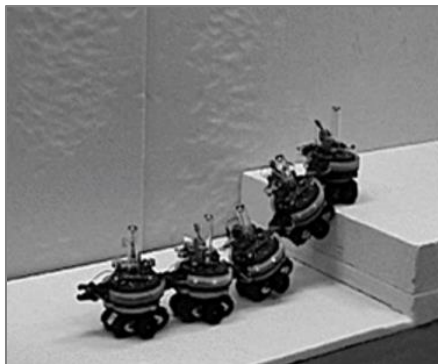


Figure 1.29: Five robots collectively tackle a 14cm step [112]

The same type of group reconfiguration is used to help the robots pass an obstacle as shown on Figure 1.29. The environments explored by robots can often be quite

unstructured or partially known. Displaying capabilities of mobility, robustness, and versatility in such situations can be done thanks to multi-robotics.

Improved performance: when tasks to be performed can be parallelized, using groups to decompose the main mission into subtasks can result in enhanced efficiency. Considering the task of transporting loads using unmanned aerial vehicles such as quadcopters, using several robots present a certain advantage for the performance of the mission (Figure 1.30).

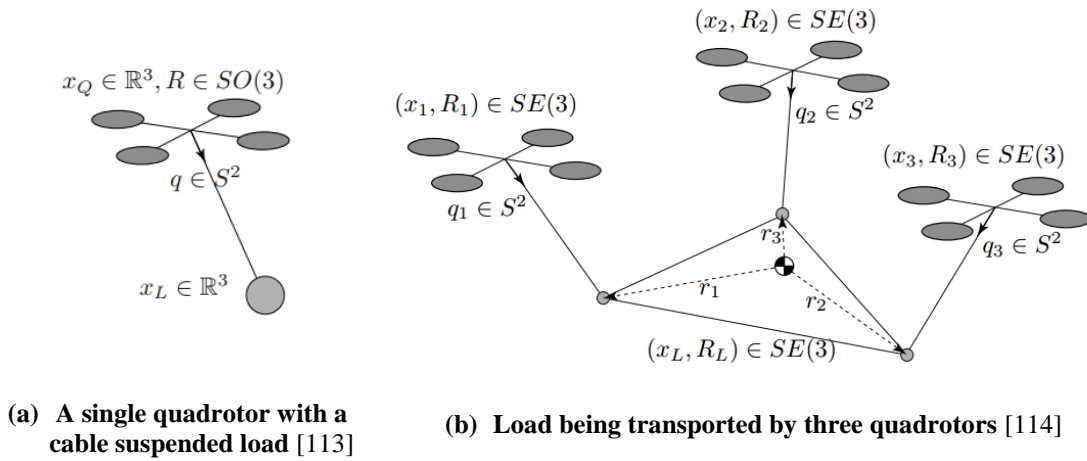


Figure 1.30: Cooperative transport using quadrotors

For a situation in which a single robot is used to carry the load, this latter will be subject to erratic accelerations and momentums that might compromise the integrity of the load. This is especially true if the load is fragile or needs to be horizontal at all times – for instance dangerous chemicals. However, by using at least three robots to carry the load, this latter can be suspended in any desired configuration [114], hence augmenting the performance of the initial system in which a single platform is used.

Distributed sensing: with multiple robots, the sensors of the system are distributed over the whole environment, hence enabling information sharing and an enhanced perception of the situation. A typical use case of this property is area surveillance: a group of drones is assigned a certain area to detect suspicious activity or detect a given target. If the task were to be performed by a single platform, this latter would have to perform a large loitering flight pattern. It would most probably miss the detection of the target, especially if the size of the surveillance area is large with respect to the resolution of the embarked detection sensor. On the other hand, if a robotic group is used, the large area is covered by several coordinated vectors, hence enabling a continuous surveillance of the whole area (Figure 1.31).

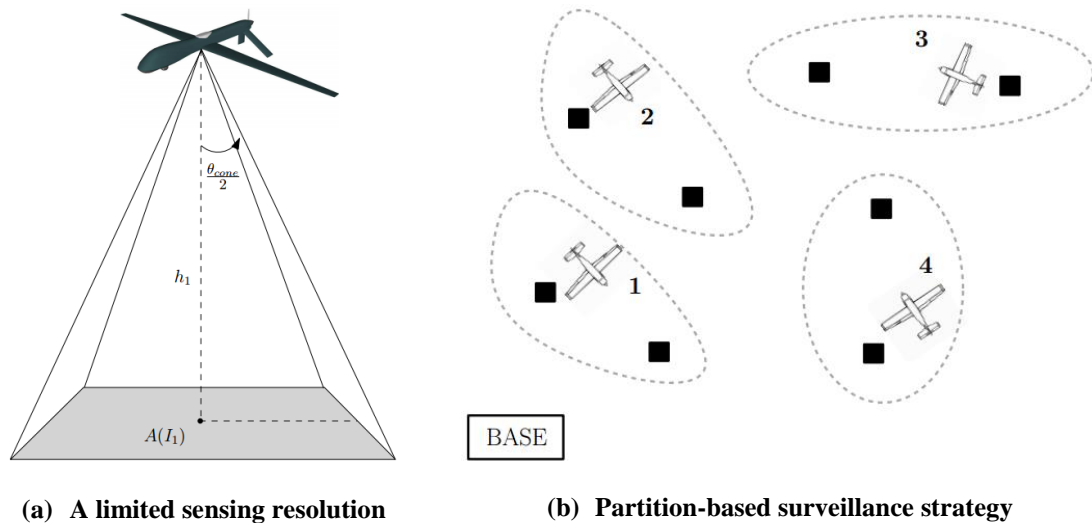


Figure 1.31: Area surveillance by UAVs [115]

Each individual drone is assigned an area optimized with respect to its own sensors and capabilities [115]. For instance, fast aircraft can cover a large area at a reasonable

resolution while quadcopter are deployed at a lower altitude, covering smaller areas with greater precision.

Another aspect of distributed sensing is that if the observations of several drones overlap, it is possible to use all of the provided information to refine the observation and increase the precision of the group. The example of mapping is explanatory: if a landmark of the environment is observed several time, each observation of the landmark is used to refine its position in the map (see Figure 1.32).

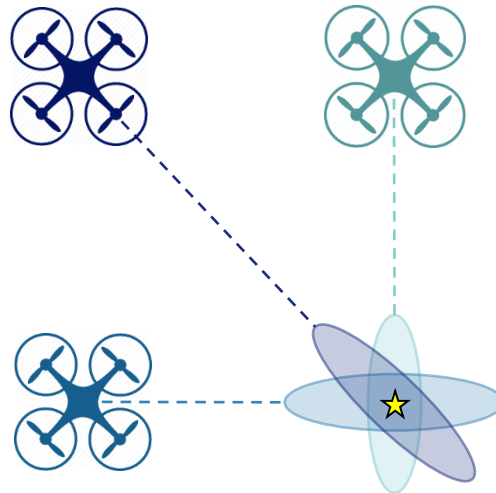


Figure 1.32: Multiple observations of a map landmark

As seen on the figure, each robot detects the landmark from a different angle with an ellipsoid of uncertainty. This latter is more important along the axis of detection due to the ambiguity in depth perception. The further the robot, the greater the uncertainty. The information provided by all observations enables to obtain a reduced uncertainty on the global position of the landmark. This may be roughly depicted by the superposition of the

ellipsoids. Each observation reduces the uncertainty along the axis of detection, resulting in a more evenly distributed location uncertainty around the landmark.

Distributed action: in the same fashion as distributed sensing, distributed action is the fact that the group of robots can perform tasks in different places simultaneously. This faculty is linked with the possible parallelism of the tasks of the mission. An example of application is bridge construction thanks to teams of robots: a group of robots can start from one end of the bridge while a matching team starts from the opposite end of the bridge (see Figure 1.33).

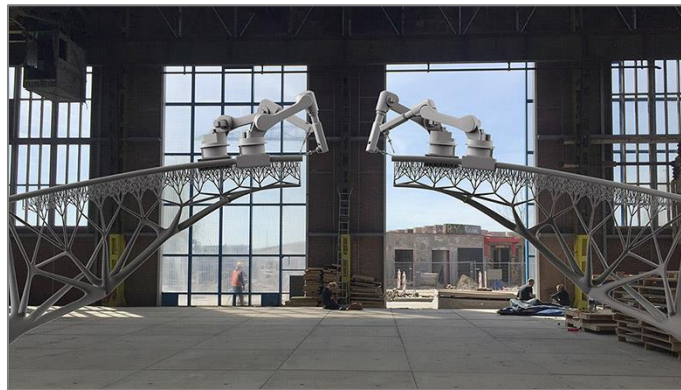


Figure 1.33: Bridge construction using distributed action [116]

Fault tolerance: having a robot failure in the group does not necessarily translate in the failure of the whole mission. Owing to the redundancy of the system, the mission can still be carried out with a limited number of robots. Using again the example of carrying a payload at a given orientation with quadrotors, this mission can still be accomplished until there are less than three robots in the group [114]. If the number drops below three, the

payload cannot be orientated anymore and supposing that the robots still have the lifting power to sustain the payload in the air, the secondary mission of simply carrying the load can still be performed. Hence, using groups of robots ensures of a certain redundancy and resiliency in the mission.

Cost reduction: the exponential growth of the UAV sector described in the previous section brings cheaper and cheaper individual units on the market. Consequently, this decreases the cost of multi-robot systems in the same fashion [33], [45], [46]. The key cost factor in multi-robotics comes from the definition: a complex goal is accomplished by simpler units. In other words, each individual platform from the group is quite incapable with respect to the overall goal of the mission. Given that simple robots are most probably cheap, this implies that for a given complex mission, a multi-robot solution can possibly be cheaper than a single-robot one. This cost difference can be illustrated with the example of aerial imagery and distributed sensing, both previously studied. If a specific area has to be imaged, satellite imagery can be obtained at cheap prices [22] but only at a given periodicity. If a group of UAVs is used for the same purpose, the original investment will be much cheaper than the initial investment for the satellite and the images can be updated very quickly depending on the mission requirements. Using plenty of cheap and low resolution sensors instead of a very expensive high resolution one, costs could hence be decreased in some cases with the concept of distributed sensing. This is the type of tradeoff study that motivates the present work.

Summary: This section showed that the possibilities unlocked thanks to multi-robotics are present under several aspects. By exploiting parallelism in the tasks of a mission, this latter can be completed more efficiently by using groups of collaborating robots. Multi-robotics also provides the ability to perform certain tasks that are impossible for single robots. In addition, using multiple robots distributes their sensors over the environment hence providing a better perception owing to information sharing. Teams of robots can also perform actions simultaneously at different locations and offer redundancy to increase the reliability of the whole system. On top of that, multi-robotics sometimes offer a cost advantage which is non-negligible at a time when projects are more and more driven under tight budget requirements. While this type of observations clearly depends on case-by-case considerations, the present research aims at providing techniques for the study of these types of tradeoffs. The next subsection details two particular real-world applications of multi-robotics to further motivate the current research.

1.3.3 Application to real-world problems

Amongst the numerous possible applications of multi-robotics mentioned earlier, two are presented with greater detail in this section. These applications of search & rescue and military operations have been chosen as they are being investigated by companies and organizations, not research laboratories.

Search & Rescue: [117] identifies that the US Coast Guard spends more than \$50M annually on search & rescue missions, a cost mainly incurred by the expensive logistics required in such operations. Search and rescue has already been performed by single robots in many occasions such as for the World Trade Center in 2001, Hurricane Katrina in 2005,

Haiti in 2010, and more recently Fukushima in 2011 [118]. As established in the previous sections, multi-robot systems and their distributed sensing capabilities offer the possibility to quickly cover search areas and detect victims much more rapidly than current methods.



Figure 1.34: Representation of a UAV-based search and rescue operation [119]

Using autonomous recharging schemes as described by [120], such systems are able to achieve extended endurance: a key asset for search and rescue missions. Moreover, the units of the multi-robot system are possibly expendable in dangerous conditions. This topic is currently being investigated for commercial applications by Saab [117].

Military operations: a study of current military systems by [121] identifies that they are highly vulnerable to coordinated attacks such as the ones that could be carried out by multi-robot systems. In particular, this study shows that any current military ship can be neutralized by a group of only eight small UAS.



Figure 1.35: Representation of a multi-UAS military system [117]

This clear vulnerability motivates the US Navy [122] and the British Army [123] to investigate the application of multi-robot systems for both offense and defense military operations. The solutions are first studied for uncluttered open water environments and will be later extended to land operations as the technologies of sense and avoid matures for trees, buildings, power lines, vehicles, and pedestrians.

Multi-robotics addresses the significant performance enhancements achievable by multi-robot systems. However, despite the increase in capability and possible real-world applications described in the previous sections, it is quite complex to comprehend, hence revealing several limitations exposed in the next subsection.

1.3.4 The limitations

The increase in capability offered by multi-robotics mostly comes at the cost of an amplified complexity as a complete System of Systems (SoS) has to be designed instead of a single robot only. According to the definitions provided in [124], a group of robots classifies as a complex system due to a large number of components able to interact with each other and the environment. In addition, the rules of the components may change over

time and are not always fully understood. Finally, the properties of the whole system cannot be determined from the simple sum of its parts. A group of robots is hence hard to design as it is larger in scope and is subject to a more complex integration of its elements. The group (macroscopic level) has to be designed as well as the individual agents (microscopic level). A non-exhaustive list of difficulties encountered when designing a system of robots are [125]:

- A higher degree of uncertainty and risk compared to classical systems design
- The system is comprised of elements with different lifecycles
- The group may be comprised of robots which are not designed to fit the whole system and which are integrated after their design is finished
- The requirements might be more ambiguous than for a single robot
- The group may have a continuous systems engineering which is never actually finished. This is especially true if new systems keep being integrated in the group in order to update it.

In addition, ground-based robots have been the norm in the field of robotics until a few years ago, limiting the interest in design optimization for robots. Subject to more stringent weight restrictions and other realities than ground robots, aerial robots design has to include optimization steps. The focus in robotics has long been on intelligence and software architecture more than on physical design. This complexity being a limitation in itself also translates into several other limitations for the use of multi-robotics.

A first drawback is interference, robots in a group can interfere with each other and hinder the proper functioning of the group. In some worse cases, it can be

counterproductive. Interference may happen with collisions, which might possibly take down the involved agents. Occlusion is also a common interference in groups of robots when one robot happens to be in the field of view of the sensors of another robot. While this can sometimes be helpful if the robots of the group have to detect each other, this occludes a part of the environment which cannot be observed. The closer the robots are, the more important the occlusion. These types of occlusion can also result in interference in communications between the vectors of the group if one robot occluding two others intercepts a message by mistake.

Another limitation given by the literature is that in some cases, uncertainty rises concerning the intentions of other robots. This is especially true for group of robots respecting exactly the assumptions of swarm robotics ensuring there is no centralized control entity for the swarm. In addition to imposing complex communication requirements on the group design, this assumption ensures that agents can only communicate with their immediate neighbors. In this case it can be hard to share the intents of other robots since there is no control overseeing the whole group. Indeed, coordination in this context requires to know what the surrounding robots are actually doing, otherwise the agents might end up competing instead of cooperating [96]. This limitation also calls for a robust decision-making architecture in the group in order to make sure that all the orders are properly given according to correct situations and not caused by miscommunication or occlusion between the robots.

Additionally, the field of collaborative robotics in general is not yet established in commercial applications and remains mostly confined to academic work such as the systems developed in [126], [127], and [128]. It is still at a preliminary stage in the research community and shy applications start to trigger interest only for the military with no known successful deployment. For instance, the Office of Naval Research (ONR) has been working on the Low-Cost UAV Swarming Technology (LOCUST) system and performed successful demonstrations in launching a swarm of nine UAVs with autonomous coordination [129]. Their next milestone is the deployment of 30 swarming Coyote UAVs from a ship for endurance missions. Although these applications are outside the academic environment, they remain confined to pure research and are not used in operation. Reasons are varied and [96] mentions in particular: the need for a good laboratory infrastructure, the difficulty in building non-linear and stochastic models of the robot groups, and the fact that currently, no general method exists to go from the individual behavior to the group behavior. [96] mentions this latter reason as a key obstacle in the elaboration of design algorithms for swarming systems. As a matter of fact, most of the research community focuses on the behavior design of multi-robot systems, leaving aside their physical design [130]. These elements prompt the formulation of another key conjecture:

Conjecture 2

A standard physical design process for multi-robot systems
is needed to foster their democratization.

Note that in the scope of this research, design corresponds to the intellectual engineering process of creating on paper a machine that either meets certain requirements and performance objectives, or explores new concepts, technologies, and innovations [131].

One additional drawback linked to multi-robotics and especially swarm robotics as defined per its assumptions, is that it tends to focus on robots having the same capabilities. As a matter of fact, one of the assumptions to exhibit “pure swarm” behavior is that the robots must be homogeneous, and that the number of subgroups with different capabilities is limited. Hence, most of the research community focuses on homogeneous groups with cheap and absolutely identical robots. Few instances such as [132], [133], [134], [135], [136], [137], or [138] consider heterogeneity in an extensive manner, and when they do, it is focused on behavior design and not physical design of the agents and the group. However, the utilization of robots with different capabilities could benefit to the mission in terms of performance and unlock new collaboration capabilities. Considering mapping and exploration missions for instance, fast robots could be used first for a preliminary pass on the environment while slower robots could later refine the initial map.

Lastly, the cost advantage mentioned in the previous sections might not always apply depending on the nature of the project of multi-robot design. This limitation does not traditionally apply since multi-robot systems aim at using many cheap simple robots which total cost still remains under the cost of complex platforms carrying out the same mission (for instance the case of DJI Phantom quadrotors versus a satellite for Earth imagery).

However, it may apply mostly when comparing single-robot systems with multi-robot solutions, and is exacerbated for heterogeneous groups. Indeed, for a heterogeneous group, several different platforms are actually designed instead of a single robot. This could result in a higher cost for the multi-robot solution in terms of development – and sometimes procurement as well. This further motivates the study of cost tradeoffs between multi-agent systems and single platform solutions.

In the presence of such limitations to the use of multi-robotics compared to its advantages, one may wonder if there is an actual benefit in developing a multi-robot system instead of a single-robot one. Indeed, most of the community tends to evaluate the performance of multi-robot systems independently without having any benchmark performance to compare it to. This is largely due to the fact that the field of swarm engineering is lacking established standards. As mentioned in [101], “Collectives offer the possibility of enhanced task performance, increased task reliability and decreased cost over more traditional robotic systems. Although they have this potential, many possible collective designs are neither more efficient, nor more reliable, nor more robust than a comparable single (more complex) robot”. In particular, it also provides a quite exhaustive analysis of situations in which multi-robot solutions would be preferable to single-robot ones. However, no quantification of the advantage or disadvantage is provided. Other studies such as [101], [133], and [134] investigate the question of whether it is more efficient to distribute expertise rather than designing a unique expert robot. However, these papers focus on very particular missions and lack generality: no methodology exists in multi-robot engineering to perform such comparisons. Similar observations can be drawn

from [139], whereas it studies the tradeoffs between having a group as opposed to a single robot, the comparison of different groups on a given mission seems to be a missing topic.

This leads to a third conjecture:

Conjecture 3

There is a need to evaluate and compare the real advantage of different optimized multi-robot systems versus optimized single-robot solutions on a given mission.

Summary: The limitations highlighted in this section showed that the design of multi-robot systems remains a quite complex problem, delaying their use in real-world applications. From SoS design obstacles to stringent communication requirements, from difficult decision-making architectures to possible development cost excess, all these complications cause multi-robot systems to struggle in establishing themselves in the commercial and even military worlds. This section also identified a need to evaluate and quantify the possible improvement or deterioration resulting from choosing a multi-robot architecture over a single-robot solution.

1.4 Summary

With an emphasis on aerial systems, this motivation section started off by describing the advantages offered by unmanned systems in terms of automation. Their growing market unleashes a multitude of unforeseen applications, hereby fostering the diversification of the robotic fleet in terms of designs and capabilities. However, this broadening spectrum of architectures is not exploited. Moreover, the advantages that unmanned systems offer might be shadowed by shortcomings in endurance and cognitive

behavior. These limitations can possibly be addressed by multi-robotics: a field inspired by nature with groups of robots performing complex missions beyond the capabilities of their simple individual components. This enables to tackle the limitations of single-system solutions by being able to accomplish more complex missions. In addition, heterogeneous groups of robots capitalize on the diversity of the current robotics fleet. Hence, it was shown that multi-robotics can be a solution to the shortcomings of single unmanned systems. Nonetheless, it was established that the advantage of multi-robot versus single-robot solutions still has to be evaluated and quantified for equal mission requirements. For instance, the tradeoff between the number of agents in a group (numerality) and the technical capability of each individual would be interesting to study. The field of collaborative robotics remains mostly confined to academia and is facing obstacles in its democratization to commercial applications. The research is at a quite preliminary stage and the applications demonstrated by the military are avant-gardist and far from deployment in real-world situations. Moreover, multi-robotics experience several limitations ranging from typical SoS design obstacles, stringent communication requirements, difficult decision-making architectures, and possible development cost excess. These complications motivate the need for a designer to quantify the possible improvement or deterioration resulting from choosing a multi-agent architecture over a single-robot solution. Moreover, the literature currently lacks methods for the designer to intelligently make these choices.

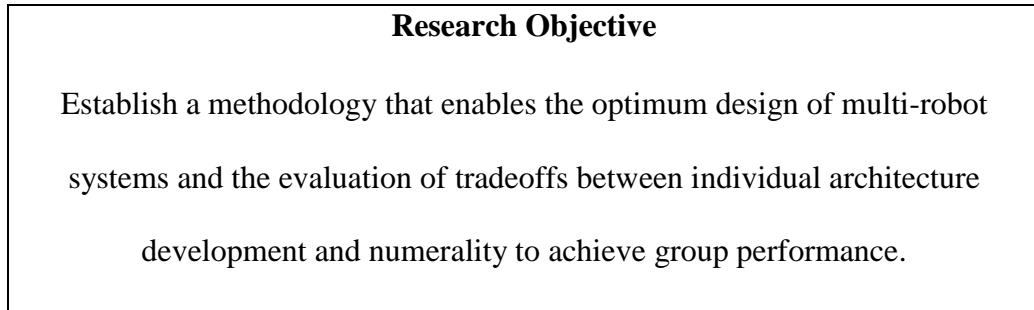
1.4.1 Research objective

The first conjecture motivates a focus of this present work on the design of multi-robot systems and especially heterogeneous groups. The second one encourages a focus on

a design process so as to increase the understanding of multi-robot systems and further democratize their use for real-world applications, beyond academic and military research. The third conjecture stimulates a research on the effects of numerosity on the group performance. In terms of design, the group numerosity could be compensated by technology enhancements on the single constituent platforms. Hence, to evaluate the real benefits of a group over single robots, the tradeoff between the number of robots in the group and the level of technological development of its agents must be studied. A motivation hence emerges with a necessity to answer some questions such as:

- Does using a multi-robot system always provide increased performance for a given mission?
- After a change in mission requirements, is it better to increase the group size or increase the capabilities of each agent in order to still be able to complete the mission?
- How to efficiently optimize groups of robots?
- Should a designer spend more time on developing the group architecture or on the Research and Development (R&D) of individual agents?
- For a given mission, what is the optimal group architecture?

These observations lead us to consider the following research objective:



The whole research process involved in establishing this research objective is outlined in Figure 1.36.

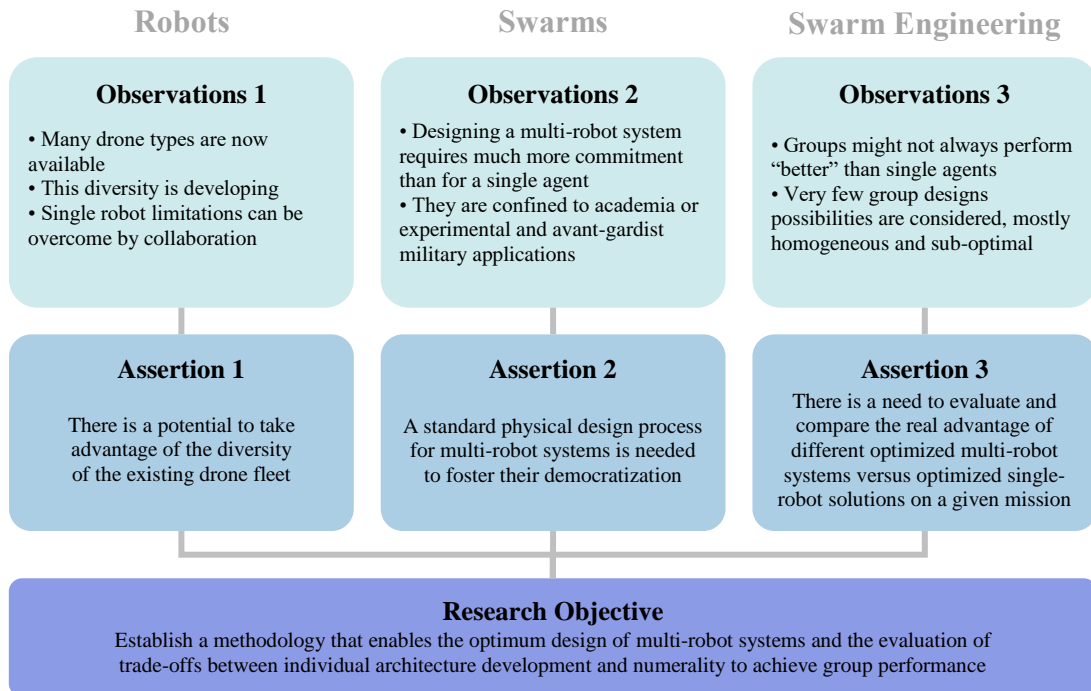


Figure 1.36: Establishment of the research objective

1.4.2 Research challenges

Reaching this objective involves addressing several research challenges owing to the novelty and complexity of multi-robot systems. The most important challenges, some of which were mentioned in the previous portions, are recollected in the present section.

First, the design optimization of a group of robots results in the generation of an extremely large design space, far beyond what is typically encountered in aerospace or automotive industries for single vehicle design for instance. A given group configuration consists of a combination of possibly different platforms, each one of them having its own design combinations in terms of subsystems and many other design dimensions. To some extent, the classical subsystems design space for single vehicles is now “multiplied” by the number of agents composing the group, this number of vehicles possibly being another design factor. This is well illustrated by considering the following toy problem:

A robotic group of three agents has to be designed from existing off-the-shelf components and architectures. Two types of robots are available, each with five subsystems and three possible technologies for each of these subsystems. Determine the size of the design space, i.e. the total number of different groups it is possible to generate from this setup. All options are compatible with each other.

Solution: the decomposition of the problem first starts with the determination of the total number of configurations for each type of robot. For five subsystems, a choice has to be made between three technologies which results in $3^5 = 243$ possible configurations for each type of robot. Then there are two types of robots available (with exactly the same architecture choices), making $243 \times 2 = 486$ the number of possible design choices for each robot of the group. Finally, it is important to note that when listing the possible combinations of robots in the group, the order does not matter. Indeed, a combination “ABC” of robots A, B and C is the same in essence as the combination “BAC” as there is one robot of each type in both combinations. Moreover, repetitions need to be accounted for since a group can be comprised of the same robots. Hence, “AAC” or “BCB” are possible combinations. The final step then consists of choosing 3 robots amongst 486 possible configurations, with repetition. From mathematics, the formula for a k-combination from a set of size n is “n multi-choose k” given by:

Equation 1.1: n multi-choose k

$$\binom{n}{k} = \binom{n+k-1}{k}$$

With $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ the usual “n choose k” binomial coefficient. The size of the design space is then $\binom{486}{3} = \binom{486+3-1}{3} = \binom{488}{3} = 19,250,136$ possible group designs. As it can be observed, a simple toy problem already generates a design space with close to twenty million possible group configurations. Using real-world figures for the number of technologies or subsystems then quickly results in extremely large design spaces. Note that

if incompatibilities between the different subsystems are taken into account, these numbers would be lower. However, all things being equal, the number of possible configurations would still remain comparatively higher than for single vehicle systems.

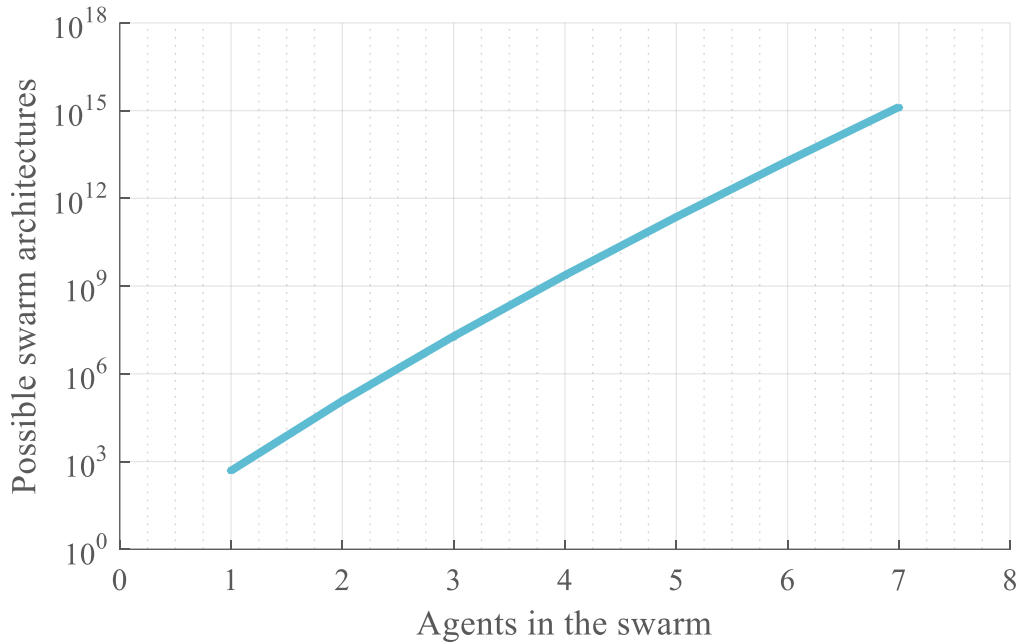


Figure 1.37: An extremely large design space

In addition, the number of possible designs grows exponentially with the parameters of this toy problem and one quickly ends up generating unmanageable design spaces which are orders of magnitude larger than for single-robot designs (see Figure 1.37). To decrease this number of alternatives, partial heterogeneity can be considered: all robots of a given architecture will have the same configuration. For instance, for a multi-robot system “ $Q_1Q_2Q_3P_1P_2$ ” composed of three quadrotors Q and two planes P , it can be assumed for simplification that all quadrotors will have the same configuration and that all

planes will have the same configuration so that the group finally reduces to “*QQQP*” with now only two configuration choices instead of five.

Moreover, the architectures to be considered in the design optimization process might also not be known a priori and have to be generated, which adds a degree of complexity to the problem. This plethora of architectures may also have elements which are not designed to be interfaced together. Hence, the compatibility of the configurations has to be accounted for in the generation of these architectures.

An additional challenge stems from the very particularity of multi-robot systems, and particularly swarming systems: their emergent behavior. Resulting from the interactions between the agents of the group and their complex cognitive comporment, this behavior is highly non-linear, possibly stochastic, and thus quite unpredictable for the design process. Owing to this non-linearity, it is in general not possible to optimize each agent individually in order to obtain an optimal swarm (see Equation 1.2).

Equation 1.2: Non-linearity of the design space

$$\sum \textit{optimized agents} \neq \textit{optimized group}$$

This distinction between additivity, non-linearity, and emergence as well as the irreducible character of group properties is clearly established by [140]. A holistic approach is required to consider the aforementioned interactions between the agents. Moreover, the lack of link between the microscopic and the macroscopic layers of a multi-robot system makes it hard to propagate design changes from the agents to the group behavior.

The design of multi-robot systems, systems of systems by definition, is also multidisciplinary. In particular, cost considerations have to be emphasized as emerging markets tend to neglect this aspect in early design phases (see Figure 1.38). Conceptual design is the first step in the design process and is an initial response to a given design goal [141]. The overall shape, size, weight, and performance of the vehicle are imagined based on basic drivers but no detailed design occurs at this stage. Essential tradeoffs are considered during the conceptual design phase.

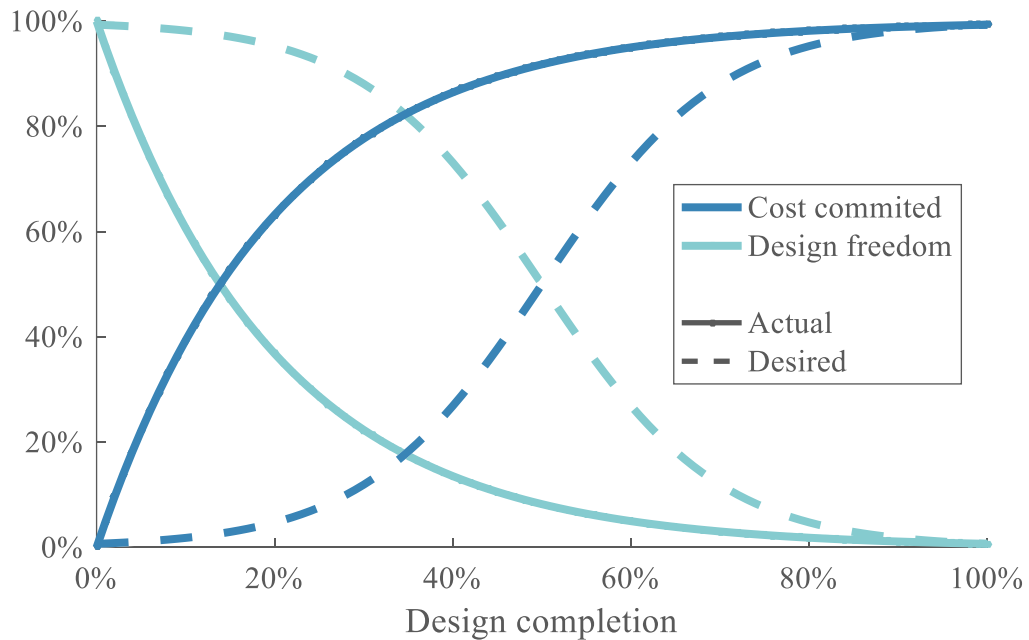


Figure 1.38: Prevalence of cost considerations in early design phases

Many disciplines have to collaborate in the design process, sometimes with conflicting impacts on the final performance, in order to create the group of robots. This

combination of technical design and financial considerations increases the difficulty of multi-robot systems design.

In addition to the challenges presented in general by systems of systems engineering, no known robotic swarm is in commercial use and the design of multi-robot systems is an emerging field. Consequently, there is a significant lack of historical data to support the design process and obtain generic insights on multi-robot systems design. There is also a lack of standards, accentuating the difficulties in establishing a clear design framework. Additionally, no benchmark is currently available, which makes it hard to quantify the outcome of the proposed research.

In order to tackle these challenges and fulfil the research objective, the assertions (Figure 1.36) must be studied individually to identify competences which already exist or need to be developed. By reviewing and comparing prevailing techniques, the next chapter pinpoints potential research gaps to be bridged. This will support the definition of the research problem and help establish the methods, tools, and processes needed to reach the goals of the research objective.

CHAPTER 2

PROBLEM DEFINITION

The path to the research objective previously established is hindered by some obstacles which must be addressed. The preceding chapter highlighted the principal challenges to overcome: exploration and optimization in a very large design space, bridging the gap from the microscopic level (the individual agents) to the macroscopic level (the group) of a multi-robot system. These form the main research focuses studied by this chapter in order to establish a proper problem definition.

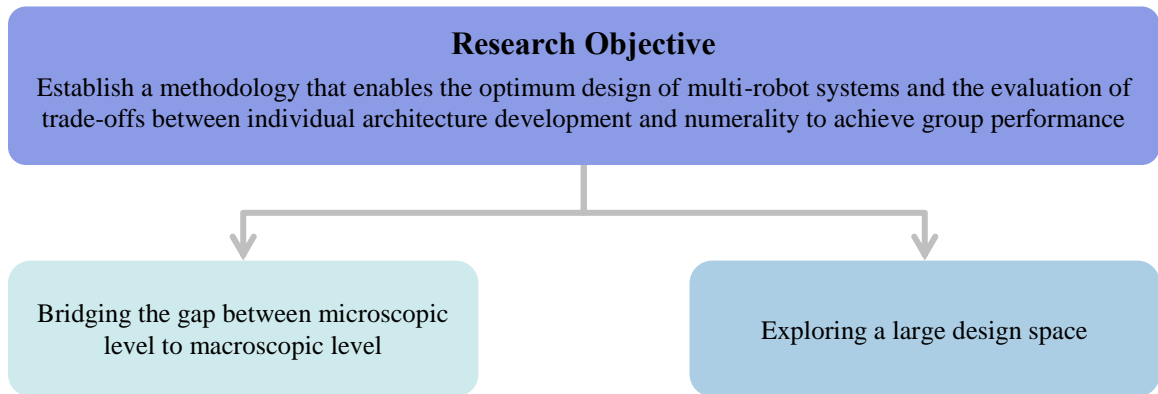


Figure 2.1: Decomposition of the research objective

In particular, a literature review of pertinent areas is proposed to understand state-of-the-art methods, their strengths but also their applicability, their assumptions, and their shortcomings. This step helps in finding the most suitable practices to address the research challenges and pinpointing deficits in the current literature through formal research

questions. Hypotheses are then developed as suggested solutions to the identified gaps with the aim of structuring the rest of the present research process.

2.1 Introductory example

With the intention of familiarizing the reader with the research objective and provide first insights on how a simulation-based approach may provide the link between macroscopic and microscopic levels of a swarm, a canonical example is formulated in this section. While this model is simplistic and corresponds to a basic 2D macroscopic model, it provides key insights shaping the research questions and the rest of this research and is to be extended with additional complexity. Consider the following problem formulation:

A homogeneous robotics swarm is to map a 2D area of size l_x by l_y .

The complete mission consists in reaching the interest area from base (distance d_0), map the area collaboratively, and return to base. One single agent has the capability to map one unit area at a time. The areas \mathcal{A}_i to be mapped by each agent i are allocated as shown on Figure 2.2. The mapping mission starts from the top left corner of each sub-area, and follows a path according to Figure 2.2, there are no obstacles to the agents. For simplicity, it is assumed that it takes one unit of time to map a unit area so that a grid cell is basically mapped once it is visited.

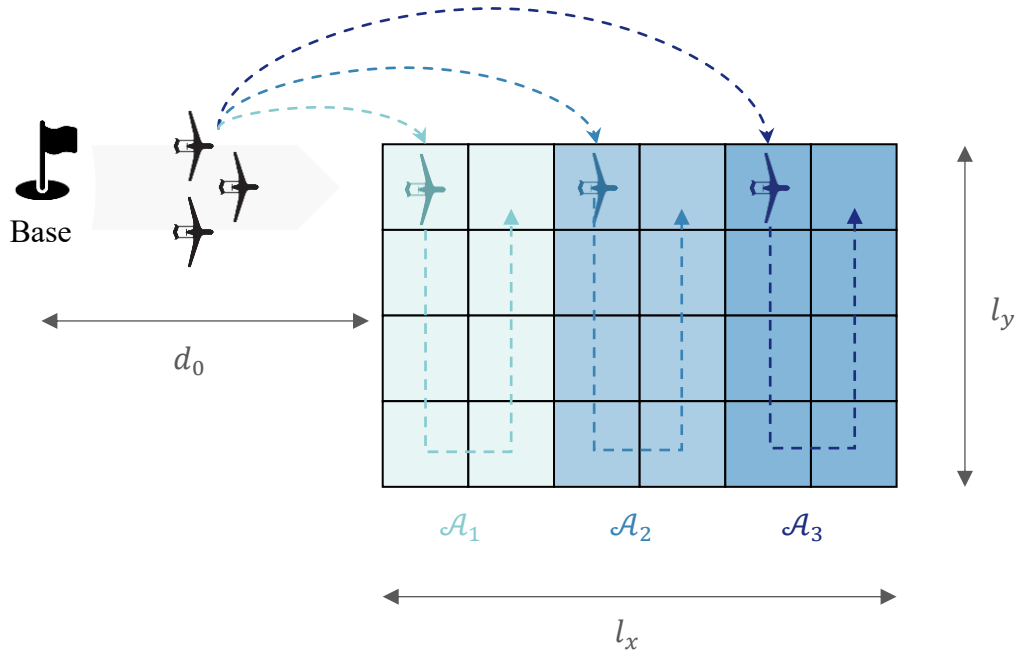


Figure 2.2: Example of mapping configuration for 3 agents

Let N denote the number of agents in the swarm and v , constant velocity of each agent, be representative of the performance of a single agent. The mission performance of the swarm is represented by T , the total time required to map the area. Each agent is assumed to have a fixed cost of c_0 , the swarm has an instalment fixed cost of C_0 for the ground station for instance. The cost of technology is represented by a linear increase c_v of the cost of an agent with the velocity. The goal is to:

1. Link the microscopic level variable v , and the macroscopic level variable N through swarm performance.
2. Assuming a maximum cost C_{\max} as a budget constraint, derive the optimum swarm configuration in terms of cost and performance.

Solution

1. The mission being finished once all agents are back at the base, the total time to carry out the mission will be dictated by the slowest agent. Note that each agent i is assigned to map an area $\mathcal{A}_i = l_{x_i}l_{y_i}$ with $l_{x_i} = \frac{l_x}{N}$. In this particular case, since all agents have the same constant velocity and the same area to map, this slowest agent is the one having to map the area the furthest from the base: agent N . Denoting t_i the time required by agent i to perform its part of the mission, this first observation is written:

Equation 2.1: Total mission time

$$T = \max_{i \in \llbracket 1, N \rrbracket} (t_i) = t_N$$

Each time t_i is then obtained by considering the distance travelled d_i . Note that a unit area is mapped in a unit of time as per the assumption stated earlier:

Equation 2.2: Mapping time for each agent

$$t_i = \frac{d_i}{v}, \forall i \in \llbracket 1, N \rrbracket$$

First, each agent has to travel the distance s_i from the base to its mapping area:

Equation 2.3: Distance from base to mapping area

$$s_i = d_0 + (i - 1)l_{x_i} = d_0 + (i - 1)\frac{l_x}{N}, \forall i \in \llbracket 1, N \rrbracket$$

Hence, it can be written as:

Equation 2.5: Distance to return to the base

$$\begin{aligned}
 b_i &= d_0 + (i - 1)l_{x_i} + \begin{cases} l_{x_i} - 1 & \text{if } l_{x_i} \text{ even} \\ \sqrt{(l_{x_i} - 1)^2 + (l_{y_i} - 1)^2} & \text{otherwise} \end{cases}, \forall i \in \llbracket 1, N \rrbracket \\
 &= \begin{cases} d_0 - 1 + i l_{x_i} & \text{if } l_{x_i} \text{ even} \\ d_0 + (i - 1)l_{x_i} + \sqrt{(l_{x_i} - 1)^2 + (l_{y_i} - 1)^2} & \text{otherwise} \end{cases}, \forall i \in \llbracket 1, N \rrbracket \\
 &= \begin{cases} d_0 - 1 + i \frac{l_x}{N} & \text{if } \frac{l_x}{N} \text{ even} \\ d_0 + (i - 1) \frac{l_x}{N} + \sqrt{\left(\frac{l_x}{N} - 1\right)^2 + (l_y - 1)^2} & \text{otherwise} \end{cases}, \forall i \in \llbracket 1, N \rrbracket
 \end{aligned}$$

Hence, the total distance travelled by an agent i is:

Equation 2.6: Total distance

$$\begin{aligned}
 d_i &= s_i + m_i + b_i, \forall i \in \llbracket 1, N \rrbracket \\
 &= \begin{cases} 2(d_0 - 1) + \frac{l_x}{N} [2i - 1 + l_y] & \text{if } \frac{l_x}{N} \text{ even} \\ 2d_0 - 1 + \frac{l_x}{N} [2(i - 1) + l_y] + \sqrt{\left(\frac{l_x}{N} - 1\right)^2 + (l_y - 1)^2} & \text{otherwise} \end{cases}
 \end{aligned}$$

Putting the result together we obtain the total time required to complete the mission as a function of the microscopic variable v and the macroscopic variable N :

Equation 2.7: Final expression for total mapping time

$$T(v, N) = t_N = \frac{d_N}{v}$$

$$= \begin{cases} \frac{1}{v} \left[2(d_0 + l_x - 1) + \frac{l_x}{N} (l_y - 1) \right] & \text{if } \frac{l_x}{N} \text{ even} \\ \frac{1}{v} \left[2(d_0 + l_x) - 1 + \frac{l_x}{N} (l_y - 2) + \sqrt{\left(\frac{l_x}{N} - 1\right)^2 + (l_y - 1)^2} \right] & \text{otherwise} \end{cases}$$

For visualization purposes, this mapping time to be minimized can be transformed into an objective to be maximized by considering the mapping rate: $R(v, N) = \frac{l_x l_y}{T(v, N)}$. This rate represents the speed at which the whole area is mapped.

2. This expression can then be plotted to gain insight on how the microscopic level and the macroscopic level variables affect the swarm performance. It may also be used to derive an optimal swarm architecture yielding the maximum performance while enforcing a cost constraint. Given the above nomenclature, the total cost for the swarm system is given as:

Equation 2.8: Cost structure

$$C = C_0 + N(c_0 + c_v \cdot v + c_{v^2} \cdot v^2)$$

The figures here below present such analysis results for $d_0 = 100 \text{ m}$ and a map size of $l_x = 100 \text{ m}$ by $l_y = 100 \text{ m}$. The cost is represented in notional units so that an agent fixed cost is $c_0 = 3$, the swarm fixed cost is $C_0 = 10$ and a unit of individual performance (velocity) increases the cost by $c_v = 1$. The quadratic technology cost factor

c_{v_2} is first set to zero. The cost constraint, or available budget, is fixed at $C_{max} = 70$ for the swarming solution. Finally, note that the mathematical formulation derived here above is valid when N is a divisor of l_x due to the mapping navigation assumptions. However, the plots are presented for various integer values of N without distinction.

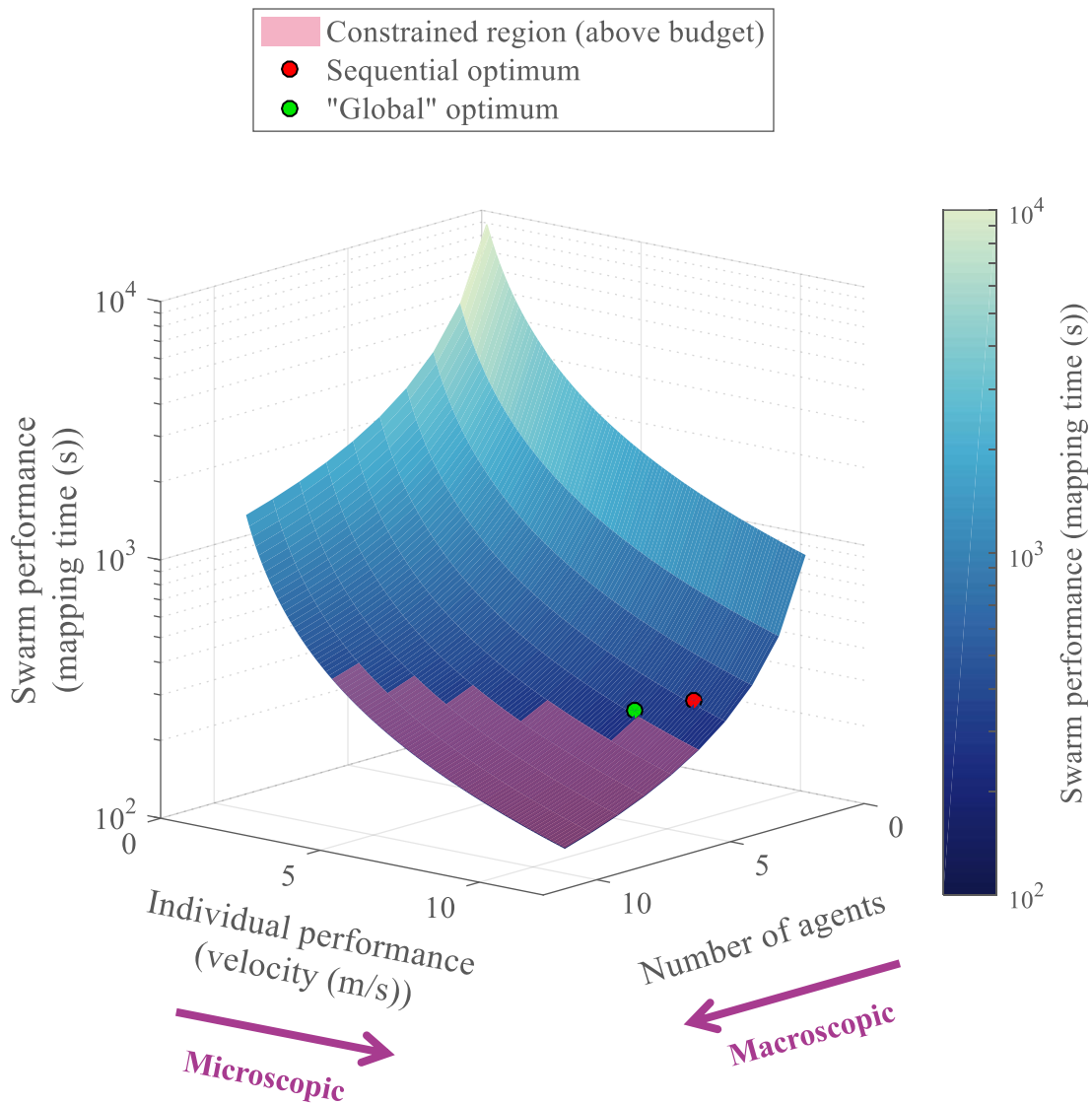
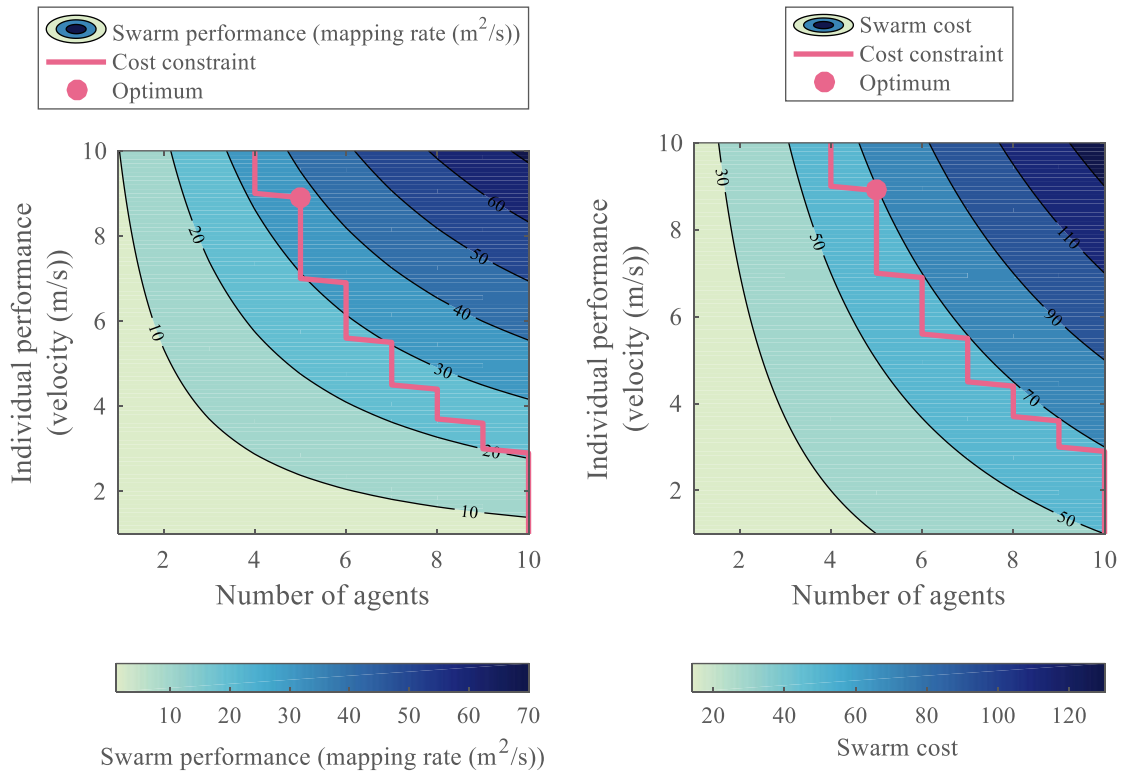


Figure 2.4: Evolution of mapping time with the design variables (example 1)

The first proposed graph (Figure 2.4) enables to understand how the microscopic level and macroscopic level variables affect the performance of the system, here represented by the mapping time $T(v, N)$. As one can expect, increasing the number of agents in the swarm increases the mapping rate, or equivalently, decreases the mapping time hence improving the mission performance. Enhancing the capabilities of each agent constituting the homogeneous swarm also results in a performance improvement.



(a) Mapping rate contours

(b) Swarm cost contours

Figure 2.5: Contours for mapping rate and system cost (example 1)

The design points which are just satisfying the cost constraint are represented in red (Figure 2.4, Figure 2.5) and they can be seen following the 70-contour of cost with a

mapping rate of $20 \text{ m}^2/\text{s}$ and $40 \text{ m}^2/\text{s}$ for the numbers given in this particular case. Optimization of the swarm design for this particular mission yields the following characteristics:

Table 2.1: Example 1 designs

Design characteristic	Sequential optimization	Global optimization
Number of agents	4	5
Individual velocity (m/s)	10	8.90
Mission time	4 min 47 s	4 min 27 s
Total cost (notional cost units)	62	69.50

Note that when deriving the optimal design with the usual sequential optimization approach (optimizing agents individually first, and then the group), a sub-optimal design is obtained (in red). By being stuck at such a local optimum, a 7% performance degradation is observed with the “global” optimization technique that optimizes both levels simultaneously. In this particular example, the optimization seems to favor individual performance over the numerality in the swarm.

By adding a quadratic velocity term to the cost with $c_{v_2} = 0.3$ and changing the cost constraint to $C_{\max} = 100$, a different optimum comparison is obtained as shown on Figure 2.6 and in Table 2.2.

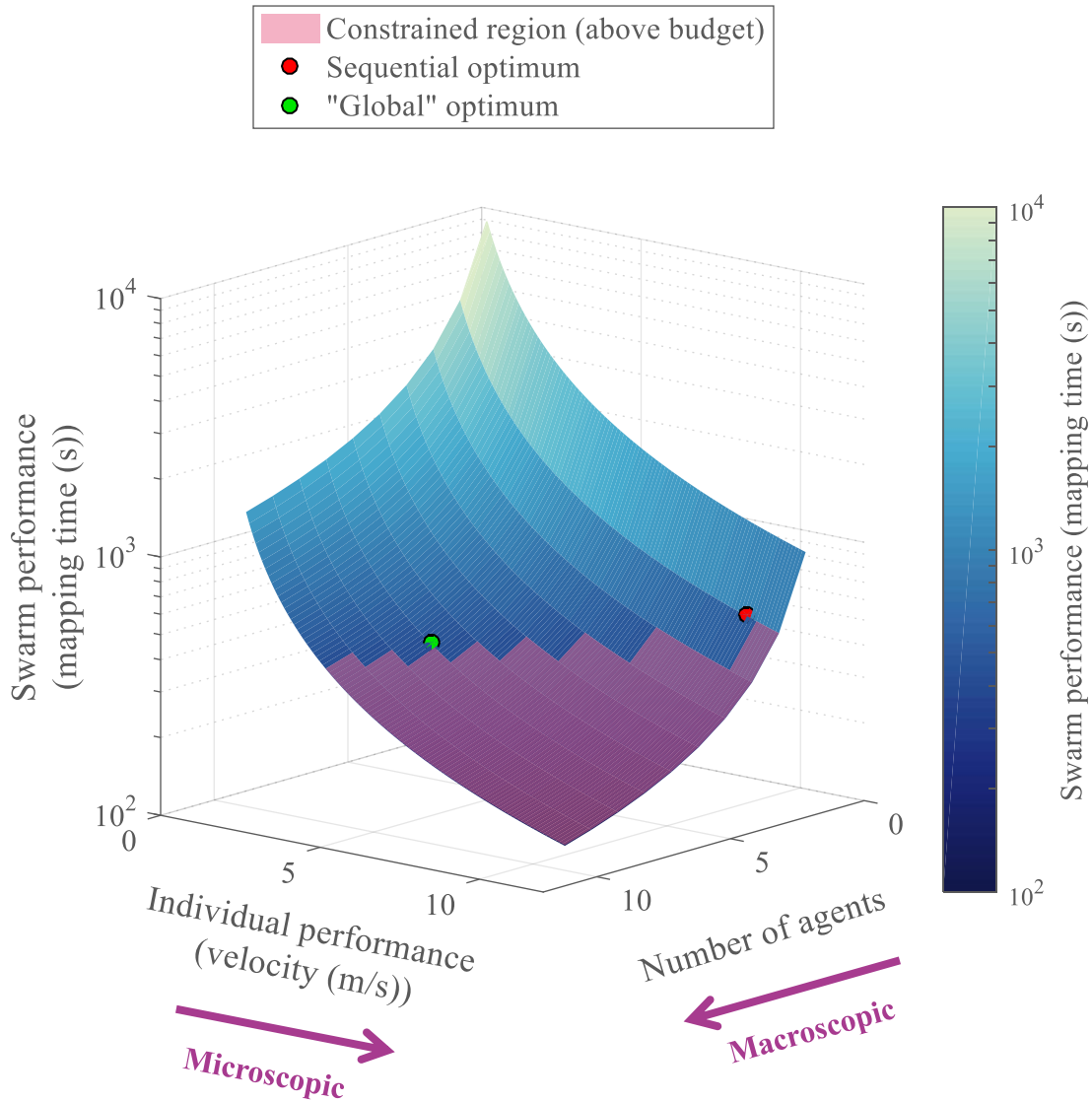


Figure 2.6: Evolution of mapping time with the design variables (example 2)

As seen on Figure 2.6, the infeasible space due to the cost constraint now occupies a larger portion of the design space. Indeed, owing to the quadratic cost term, the cost of the system now increases more importantly with the velocity design variable.

Table 2.2: Example 2 designs

Design characteristic	Sequential optimization	Global optimization
Number of agents	2	7
Individual velocity (m/s)	10	4.30
Mission time	8 min 55 s	7 min 01 s
Total cost (notional cost units)	96	99.93

This time, the optimization tends to favor numerality over individual performance by having many agents with a moderate velocity. One will note that the simultaneous optimization technique offers an improvement of 27% in performance compared to the sequential optimization scheme.

This example helps illustrate and understand the idea of tradeoffs between numerality and individual performance, main component of this research. The agents are modeled with one variable and the group behavior is fixed (although depending on the number of agents and the size of the map) and is not overcomplicated with respect to the complexity of the microscopic level. This is the type of tradeoff that is used in mesoscopic models, detailed in the next subsection. The purpose of this present work is then to build from this simplistic case, notably by adding additional levels of complexity: heterogeneity, agent behavior, swarm intelligence, and possibly stochasticity and uncertainty.

2.2 Bridging the gap from microscopic to macroscopic level

The first principal research axis identified in this problem formulation is the lack of a link between the behavior of the agents and the compartment of the group (Equation 1.2). This existing gap must be bridged in order to be able to obtain optimal group designs.

The emergent field of swarm engineering tries to address this design difficulty of multi-robot systems and to standardize their design. Introduced by [142], the term of swarm engineering refers to the emergent discipline aimed at forming systematic processes to model, design, realize, verify, validate, operate, and maintain swarm robotics systems [95] and to a larger extent, multi-robot systems. The formal method of swarm engineering as defined by [142] consists of two steps: the generation of an appropriate group-based swarm condition, and the generation of a behavior for each swarm agent that satisfies this condition. Considering swarm engineering in a broader sense, the formalism of systems of systems engineering provides a first approach to understand what tools and methods are required in order to design a group of robots. Swarm engineering being a relatively new field, it exhibits a certain lack of maturity which adds many limitations to its current possibilities. In this section, a review of these restraints is proposed and by focusing on a few of them, research questions are introduced. A multitude of specific swarm design methodologies are then reviewed to gain depth in understanding swarm engineering and possibly give first elements of response to the research questions.

2.2.1 Swarm engineering: a lack of maturity

It is not before the year 2000 that a first design procedure focused on multi-robot systems and more particularly swarming systems was formalized by [142]. This makes the field of swarm engineering quite recent and at a very early stage of its development. Subsequently, this emergent discipline experiences several shortcomings which hampers its understanding and delays its usage in civil applications.

To begin with, the development of swarm engineering is not homogeneous. The areas of requirements analysis, performance measurement and maintenance are clearly lagging behind more popular topics such as design and analysis which concentrate most of the progress of the field [95]. Even in design and analysis, much emphasis is put on the behavior of the individual agents and the behavior of the swarm, giving little attention to the physical design of the robots [130]. Most of the time, a generic swarm of robots is built focusing on the simplicity of the agents and their means of local interactions (cameras and markers, infrared sensors, Wi-Fi...).

Once the swarm is built, the research focuses mainly on the implementation of new behaviors for the swarm. It is likely that once swarming systems are introduced into real-world applications, maintenance and performance areas will start being a concern for the industry and this interest will transfer to the research community.

The development of design approaches is also held back by a lack of established standard metrics to evaluate the performance of robot swarms. Few metrics are defined and most of them are tightly related to specific applications and cannot be reused. For instance, when considering a mapping and exploration application, relevant metrics can be the area covered, the quality of the map, and the time to cover this area. However for construction or pattern formation applications, such metrics would not make a lot of sense. This particularity makes the design of swarming systems very application-specific, preventing the establishment of a generic design method.

Additionally, the field is missing testbed applications and publicly available datasets for the research community. These are essentials in order to test and benchmark new design methods and algorithms. While foraging is commonly used as a test application, it is limited and lacks an established standard scenario [95] that could be used by researchers. Construction, also commonly used as a testbed, exhibits similar limitations. This lack of standards hinders the comparison of different swarm robotics systems.

Swarm engineering is also impeded by the absence of an established simulator which could be an enabler for the research community. An ideal simulator for multi-robot purposes should first encompass features from typical robotics simulators: enable 3D simulations and be modular to accommodate any type of robots. It should also be scalable with respect to the number of robots. This is the main constraint for multi-robot simulators as swarm robotics in particular might require an extremely high number of agents. In addition, modularity is essential to be able to accommodate different scenarios. Finally, the ideal swarm engineering simulator should be open-source to foster development by the research community. Many mobile robotic simulators already exist and may be used for experiments with multi-robot systems. Their differences lie mainly in their technical characteristics but also the cost of their license [96]. These principal suites of simulators as reviewed by [96] and [143] are summarized in Table 2.3 with their capabilities.

Table 2.3: Overview of multi-robot simulators

Simulator	3D	Scalability	Modularity	Open-source
ARIA		●●	●	YES
CARMEN			●●●	YES
Gazebo	●●●	●	●●●	YES
Microsoft Robotics Studio	●●●	●	●●●	NO
MissionLab	●●●	●		YES
Pyro	●●●		●●	YES
Stage		●●●		YES
SwarmBot3D	●●●	●●●		YES
TeamBots		●		YES
Webots	●●●	●	●●	NO

The legend used for this table is the following:

- Feature may be obtained by extension
- Feature partially/decently supported
- Feature completely supported

The development environments which are the most used include Stage which provides 2D simulation capabilities with a runtime scaling mostly linearly with the number of robots up to 100,000. Gazebo is a 3D open-source simulator which notably includes rigid-body physics and comprises a large library of robots and sensors. Webots is a commercial mobile robotics simulator providing the same features and enabling multi-robot simulations for up to 100 robots. Microsoft Robotics Studio is another simulator developed by Microsoft Corporation. It enables multi-robot simulation but suffers from discontinued official support since 2012. Finally, SwarmBot3D is a simulator designed

specifically for the S-Bot platform of the SwarmBot project. It can be seen that none of these simulator offers all the characteristics required for swarm engineering simulations. As Table 2.3 also suggests, no existing solution proposes to address all the enabling capabilities of the ideal multi-robot simulator.

Furthermore, highly non-linear relations from agent parameters to group behavior [144] result in a lack of top-down design methods [95] and establish a gap in the design flow. Indeed, as stated in [145], “one challenge in understanding self-organizing systems is the micro–macro link, i.e., determining the relationship between global and local behavior patterns and vice versa”. [96] also states that “no general method exists to go from the individuals to the group behavior”. Due to the multitude of concurrent interactions among multiple agents, the non-linear relationship between the variation in a design parameter and its resultant repercussion on the group behavior is non-intuitive and hard to control [144].

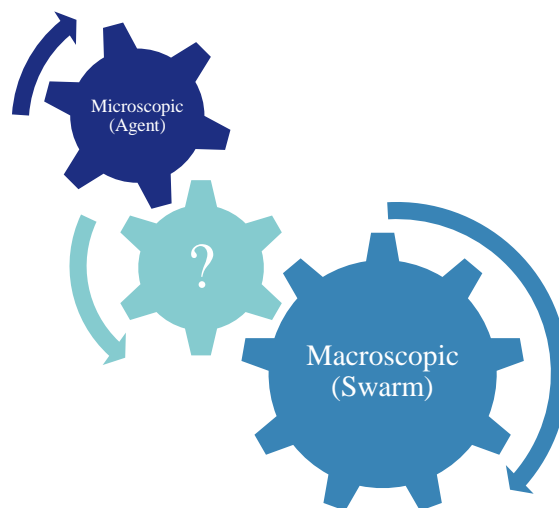


Figure 2.7: Missing link between macroscopic and microscopic level

This lack of a link between macroscopic and microscopic levels of a swarm is the main obstacle faced by swarm engineering and the intuition of the human designer still remains the main ingredient in the development of swarming systems [95].

Optimization of the different attributes of the group and its individual agents is part of the traditional design process and the manifestation of this gap is a sequential optimization with suboptimal results. This deficiency results in current methods of swarm engineering tackling the design of swarming systems by performing a serialized optimization: the microscopic level first (the agents), and then the macroscopic level (the swarm). As a matter of fact, some of the research community also use in their swarms existing platforms which are already optimized for their respective and completely different missions. Amongst many other examples, [133] has the system designer creating multi-robot systems by picking from a pool of available robots. [146] alters pre-existing platforms to obtain a multi-robot system suited to the considered mission. [147] prepares a swarming system for a competition using the most common quadcopter for the general public: the A.R. Parrot Drone. These missions generally greatly differ in scope with the type of mission that the swarm has to accomplish. Using these platforms, researchers then attempt to obtain an optimal swarm by completing a second round of optimization. This sequential process results in suboptimal designs since on one hand, it is unlikely that the swarm takes advantage of the full capacities of its constituent agents, and on the other hand the agents were never designed fully as the integral part of a swarm. In his study for architecting systems, [125] is one of the first to highlight this incoherence in the design process back in the early days of systems of systems engineering. Focusing on communication aspects, it is stated that the resulting multi-robot system might be

comprised of agents which were not designed to fit the whole system. They are integrated after their own design is finished.

While the lack of this micro-macro link is identified by many in the research community [95], [96], [144], [145], there is no record of an attempt to quantify the possible improvements which could be achieved in design performance if such a link was to be established. This observation leads to the first research question:

Research question 1

Can multi-robot systems designs be improved by linking
microscopic and macroscopic levels?

Managing to link microscopic and macroscopic levels of a swarm in terms of design optimization would consist in replacing the sequential optimization method with what is generally referred to as a more “global” or “simultaneous” optimization approach. Hence, to articulate a hypothesis regarding research question 1, the differences between sequential optimization and global optimization are reviewed in detail.

This segregation between sequential and global optimization typically appears in Multidisciplinary Design Optimization (MDO), a field used in the design of complex systems. Swarms constitute a good example of such systems as they are “an assembly of interacting members that is difficult to understand as a whole” [148]. Moreover, they are characterized by an emergent behavior, a large number of design variables, and the nonlinearity of the individual interactions. With the intention of coherently exploiting the

synergism of these mutually interacting components, MDO partitions the problem into subsystems based on physical boundaries, functionality, disciplines, or even organizational structure of a company. For the scope of this work on group robotics, the system is decomposed based on physical boundaries: the different robots composing the swarm. MDO problems are different from classical optimization problems as the design variables feed into multiple analyses, the objective function depends on several analysis outputs, and interdependencies might exist between the different analysis functions.

Sequential optimization involves optimizing the different constituents of a system independently and sequentially, with their own set of constraints and cost function. For a robotic swarm for instance, it means that the agents of the swarm are first optimized with respect to individual mission requirements, the whole swarm is then optimized using this set of pre-optimized agents, or vice-versa (Figure 2.8). On the other hand, global optimization or simultaneous optimization tends to consider all constituents at once with all their interactions, in order to derive an optimal complete system (Figure 2.9).

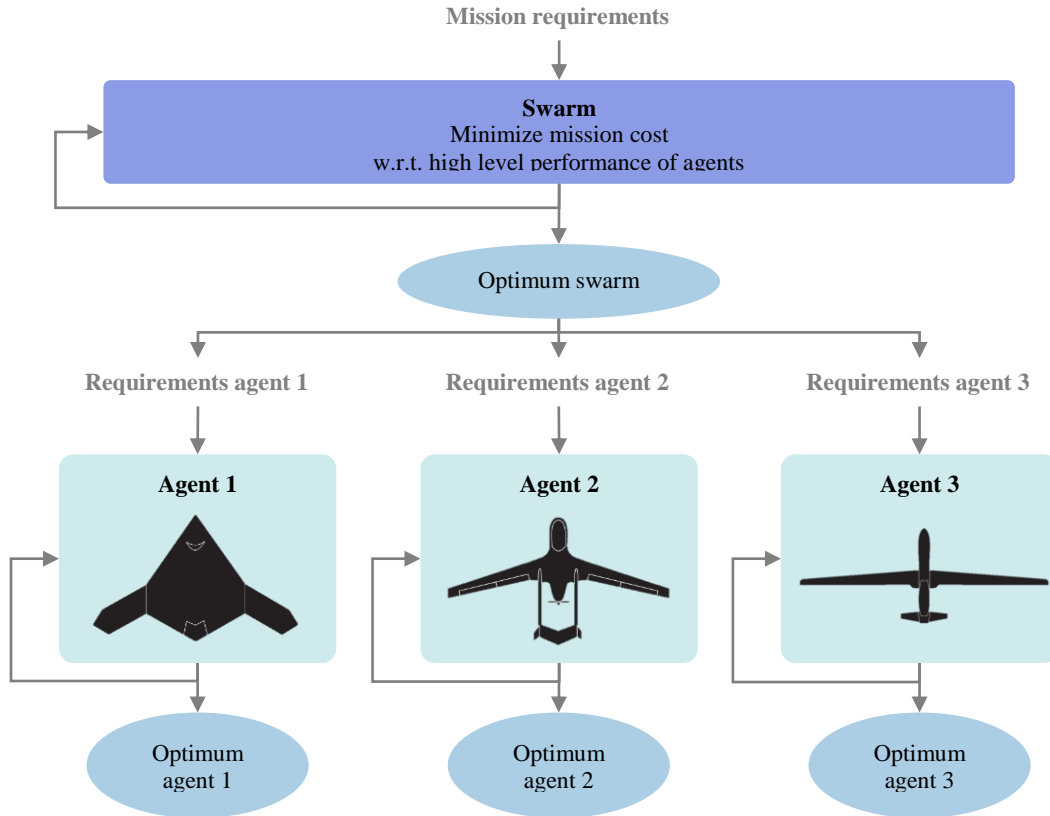


Figure 2.8: Example of sequential swarm design optimization

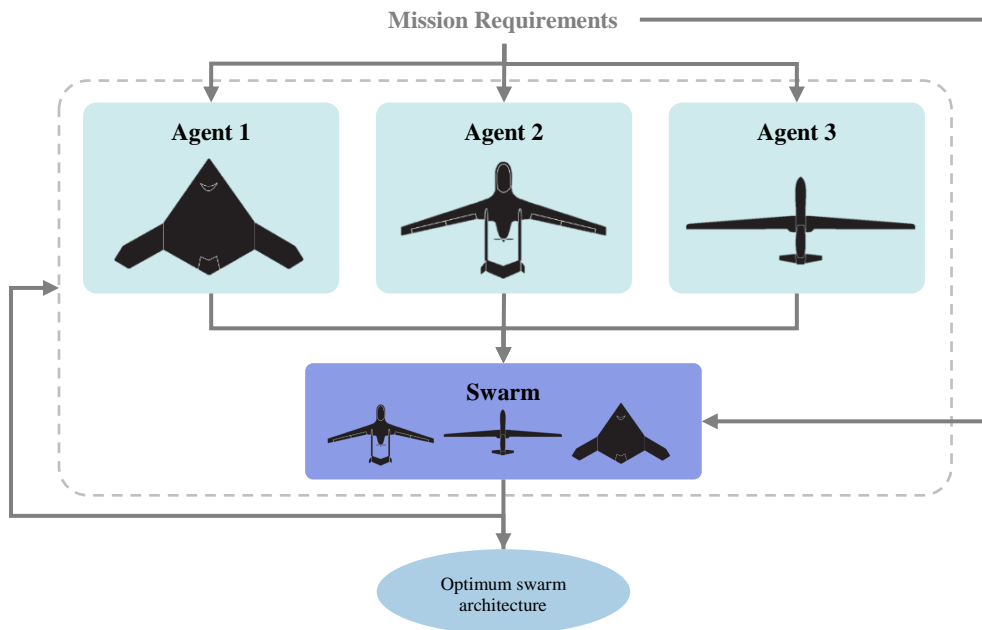


Figure 2.9: Example of global swarm design optimization

A growing number of methods of MDO are available to perform global optimization and they can be divided into two classes: monolithic or single-level formulations, and multilevel formulations [149]. Single-level techniques use a single optimizer at the system level and distribute the analysis to the partitioned subsystems. Such methods tend to be easy to implement but they might not scale well for industrial settings and for very large problems with many disciplines. On the other hand, multilevel methods use an optimizer at the system level in addition to multiple other optimizers at the subsystem level. By benchmarking different MDO algorithms, [149] clearly states that strategies such as sequential optimization are often not able to find the true optimum of a system and it is underlined that interactions between the components of the system must be properly accounted for. This last statement is presumed to be true for multi-robot systems, and the previous introductory example offers evidence of this suboptimal behavior. One goal of this thesis is to examine this widely held assumption when applied to multi-robot systems.

This statement is corroborated by [150] on an example of aerospace design: this partial (sequential) optimization approach does not lead to the optimal design of the complete system, except in special cases. On Figure 2.10, a wing is to be designed in order to achieve a minimum value of an aggregate function of weight and drag. The system optimizer acts on the span which is given to the aerodynamics group in charge of minimizing the drag by determining an optimal twist distribution. The aerodynamics loads acting on the wing are then forwarded to the structures group, in charge of minimizing the weight of the wing with respect to the skin thickness.

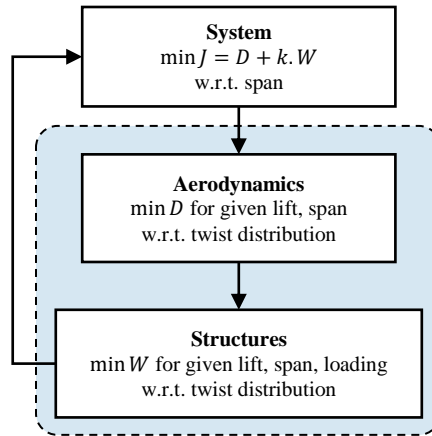


Figure 2.10: Wing design by sequential disciplinary optimization

This process is iterated for different values of the wing span until an “optimum” design is reached. It is shown that this type of sequential procedure yields 11% more drag than an optimal solution derived with a global optimization technique [150]. Similar conclusions are drawn from aeroelastic and dynamics considerations. This type of optimization procedure is notably used in [151] where a bi-level algorithm is used: one level dedicated to the design characteristics and high-level variables (material, curvature, global thickness, struts...), and a second level reserved for design proportions (skin thickness, cross-sections...).

Reference [152] also constated the possible improvements achievable through simultaneous optimization by applying it to suborbital vehicles design programs. It was shown that programs that do not properly account for the interactions between their different business divisions (hence using sequential optimization schemes) proposed very different programs than the optimal ones found by simultaneous optimization. An

improvement of up to 14.1% was announced as a result of using a global optimization scheme as opposed to a sequential one.

The difference between sequential and global optimization is also well illustrated in the field of industrial engineering, and particularly for supply chain management problematics. In such problems, a system of suppliers, manufacturers, transportation, distributors, and vendors operate to transform raw materials to final products and supply those products to customers. Optimization is applied so that each component of the chain orders its necessary input in the right quantities, in order to be able to deliver the expected output at the right time. The goal of supply chain management is then to minimize the cost of the complete system despite conflicting objectives from the different facilities at stake [153]. The most evident of these tradeoffs is that the customer desires a short time to delivery at low prices while the warehouses focus on having low inventory to reduce their operating costs. In this context, [154] compares sequential optimization to global optimization on a simplified supply chain. In the first case, the agents of the chain derive their respective optimal solution independently of the others whereas a complete integrated supply chain system is considered for the second case (Figure 2.11).

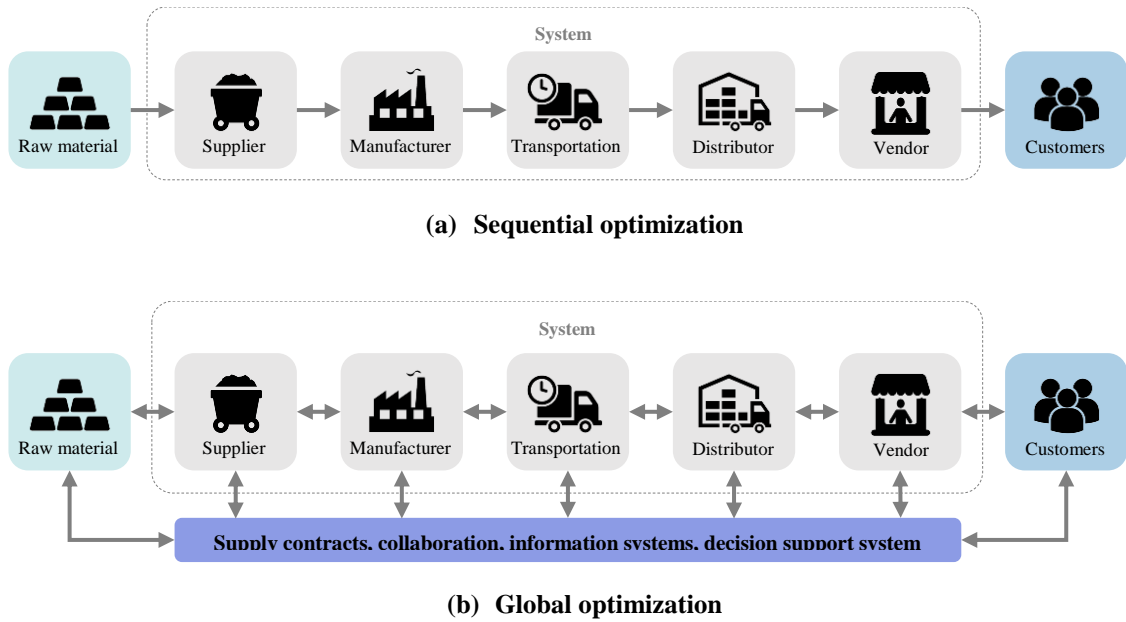


Figure 2.11: Optimization in supply chain management

Hence in sequential management, each party optimizes its own profit with almost no regard to how its decisions affect the other members of the supply chain. On the other hand, a global optimization scheme tries to find what is best for the entire supply chain. Although the global optimization scheme is harder to put in place for supply chain management since all parties have to agree to share their inventory information, it results in better performance of the overall system than in the sequential optimization case. A total joint profit increase of 2.31% is demonstrated in the example of [154] and the results show that the global optimization scheme is better than the sequential one.

In the light of this review of MDO, it appears that if a link can be formulated between microscopic level and macroscopic level, global optimization schemes can be used to optimize the design of a swarm, resulting in an improved performance when

compared to the sequential optimization case. This can be formulated as the following hypothesis:

Hypothesis 1

IF an approach leveraging the interdependence between
microscopic and macroscopic levels is used
THEN significant improvements in average performance can be
achieved in the design of multi-robot systems compared to
traditional sequential optimization schemes.

To validate this hypothesis, experiment 1 is implemented to compare the performance of different multi-robot system designs on the same testbed mission. In particular, typical questions to be answered by this experiment are:

- Can the micro-macro link provide clear benefits in multi-robot systems design?
- How much improvement can be obtained? To a greater extent, how do the two methods compare?
- In which cases are both methods yielding similar results?

The first swarm design is obtained with a sequential optimization while the second design is derived through global optimization. This experiment is to be repeated for several swarm architecture choices in order to obtain clear conclusions. The necessary requirements for such an experiment are detailed here below:

- Testbed mission: in this case a mapping/imaging mission (see section 3.3.4)

- Method to link microscopic and macroscopic level
- Sequential optimization algorithms
- Corresponding global optimization algorithm: for instance if a genetic algorithm is used for the sequential optimization, then the same type of algorithm must be used in the global optimization scheme.

Metrics to compare the two methods are also required in terms of number of iterations, number of objective function evaluations, rapidity, stability, precision, and other relevant optimization metrics (see [155]). Based on the questions to be answered, the evaluation criteria for this experiment are: the percentage difference in the optimization function values between sequential and global optimization cases, the proportion of test cases where the difference is below a given limit as well as other insightful statistical measures of the experiment. The optimal group performance will be assessed with respect to the best known solution since no analytical derivation of the true optimum is feasible [156].

The test cases involve first optimizing a set of architectures independently for a testbed mission, and then optimizing a swarm composition based on these architectures for the testbed mission. The second step is to perform the simultaneous optimization of both the architectures and the swarm composition before comparing the two methods. This comparison is to be repeated for different architectures and mission characteristics. In particular, additional parameters which could be varied during the experiment may include the problem size, the stopping criteria of the optimization procedures, their search

neighborhoods, and the move selection. A failure point for this experiment consists in the case where both methods perform equally well or even when the global optimization method performs statistically worse than the sequential approach. The first validation criterion for hypothesis 1 is that based on statistical testing, global optimization yields a superior main design performance on the testbed mission at a 95% confidence level. The second criterion is that the computational time required is comparable to sequential optimization. In particular, for the proposed methodology to be used and adopted by designers, it is necessary to ensure that the global optimization procedure is not several orders of magnitude slower than the sequential optimization procedure currently used in the field.

This experiment is designed to provide quantifiable response elements to the widespread claim that the absence of micro-macro link results in sub-optimal swarm designs. However, to be able to carry out these experiments, at least one method to link the microscopic level and the macroscopic level of the swarm has to be elaborated with an emphasis on physical design. Focusing on conceptual design, central effort for this research, another research question directly derives from the first research question:

Research question 2

How to link the microscopic and the macroscopic levels of a multi-robot system for conceptual design purposes?

In order to bring elements of response and formulate a hypothesis, the next section examines a multitude of possible design methods which could be utilized to link the microscopic and the macroscopic levels of a swarm.

2.2.2 A diversity of design methods

Swarms include some additional particularities (see 1.3.1) which require dedicated techniques for their design. Such design and analysis techniques fall under the definition of swarm engineering as per [142] and are presented in this section based on the taxonomies proposed by [95] (see Figure 2.12).

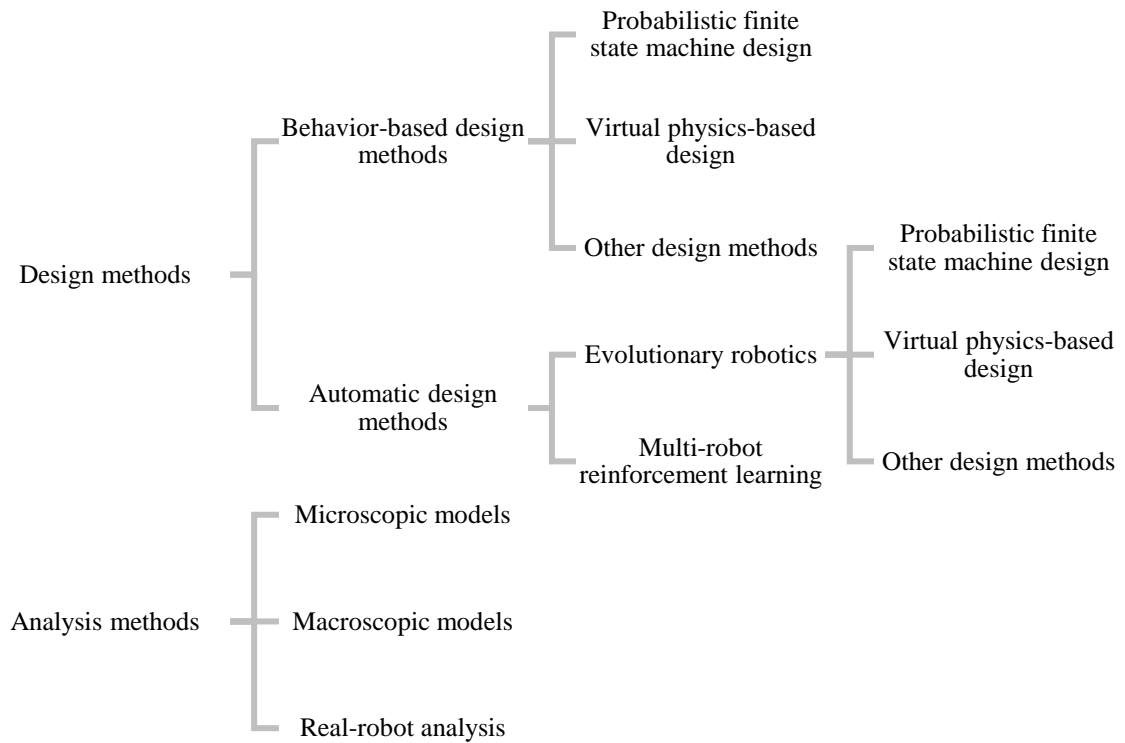


Figure 2.12: Swarm engineering methods taxonomy [95]

The next two subsections examine both the design and analysis methods as they represent possibilities in solving the micro-macro link problem. In particular, it is shown that none of these methods currently answers exactly the research question envisaged in the scope of this work. Design methods focus mainly on the design of behaviors as opposed to physical design, and the analysis methods are completely uncoupled or unfeasible, hence lacking of a link between microscopic level and macroscopic level of a swarm.

2.2.2.1 Design methods

The goal of the design steps is to create a swarm in terms of architecture, its vehicles, as well as its functioning so that the group meets certain criteria or accomplishes a given mission based on initial requirements. The aim of this section is to show that despite an apparently high number of design methods, swarm engineering actually focuses mostly on the behavioral design of the swarm, leaving aside the physical design of the agents, main focus of this present work.

2.2.2.1.1 System of systems approach

The unique engineering challenges presented by robotic swarms seem to correspond to the ones addressed by systems of systems engineering. Indeed, the goal in designing a robotics swarm is to optimize the design of the swarm itself but also of its agents individually so that they fit perfectly into the group. This section will present a typical SoS approach to show how this field can be used to design multi-robot systems and possibly swarm systems. A tentative definition of systems of systems given by [157] is:

Groups of systems, each of which individually provides its own mission capability, that can be operated collectively to achieve an independent, and usually larger, common mission capability.

This designation is to some extent reminiscent of the characterization given for swarm robotics, hence consolidating a link between SoS engineering and swarm engineering. The field of SoS engineering is an arising interdisciplinary methodology focused on transforming individual capabilities into system of systems solutions [158]. In particular, SoS engineering ensures that individual systems function as autonomous constituents of one or more SoS, providing appropriate functional capabilities. The management and operations part of SoS makes sure that political, financial, legal, technical, social, operational, and organizational factors are considered. This includes the perspectives and relationships of the stakeholders.

The main challenges presented by [159] which are of main concern for system of systems engineering are:

- Complexity is a major issue
- Management can overshadow engineering
- Initial requirements are likely to be ambiguous
- System elements operate independently
- Fuzzy boundaries cause confusion
- System elements have different life cycles
- SoS engineering is never finished

The individual systems are not necessarily designed with the goals of the whole SoS in mind and the field also has to focus on the flow of data and resources between systems that were most probably not designed to be interfaced. Moreover, testing and evaluation of the overall system can prove difficult owing to the vague performance criteria. Finally, system of systems engineering also emphasizes the importance of political aspects of the design process. Indeed, system of systems are generally funded by a broad range of sources due to the scope of such projects, making it difficult to align the goals of each individual funding party with the overall SoS purpose.

In terms of design methods, SoS engineering first concentrates a large effort on “architecting” [158], [160]: describing the fundamental organization of the system and its components as well as their relationships to each other and the environment. The architecture is usually established using a framework which defines specifications for each “view”. Each view offers a unique perspective about the system and can be focused on management or integration for instance. The main advantage of this step is to provide a mean to organize, communicate, and document details about the systems, establish a planning, and detect possible pitfalls in early designs. Key products of this phase are the identification of requirements for the modeling and simulation step, a means of communication between stakeholders and engineers and an evaluation of risks and costs.

Modeling and simulation is then used as a second principal step to estimate performance metrics and evaluate the goodness of a design alternative. The views previously created help provide details needed for modeling the chosen architecture. The

link between the products of an architecture and a dynamic modeling and simulation software is often established through an “executable architecture” [158]. To enable executable architecting, the products from the architecture must be standardized, computer-readable, consistent, and contain the information required by the modeling tools. Figure 2.13 summarizes the design process used in architecture-based SoS engineering.

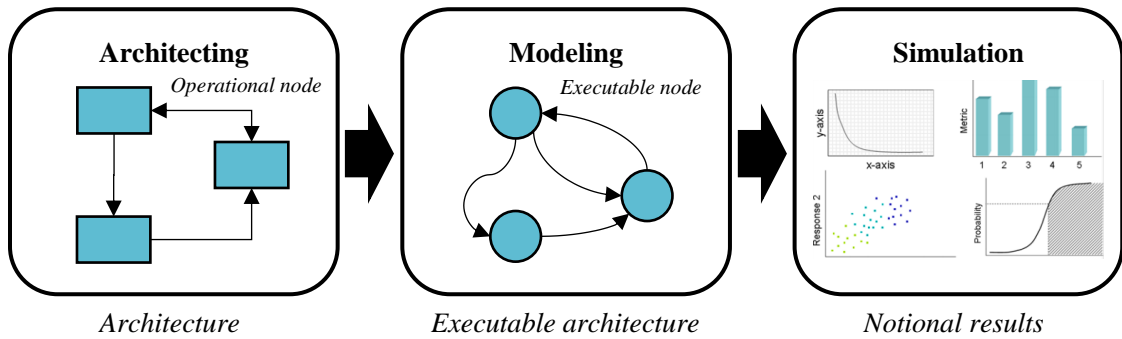


Figure 2.13: Architecture-based SoS engineering process

Modeling for SoS can exhibit some challenges due to the complexity and emergent behavior required for the mission to accomplish, the stochastic and dynamic nature of the requirements, and non-linearity.

Surrogate modeling may be used in the process and presents its own challenges since as it was stated previously, the relation and propagation from individual agent parameter (microscopic level input) to the overall swarm behavior (macroscopic level output) is highly non-linear. Moreover, designers are very often interested in having a time-dependent description of the behavior of the swarm rather than a non-dynamic model. Hence, particular advanced surrogate modeling techniques and their associated Design of Experiments (DoE) procedures are used for SoS engineering. Such SoS modeling

techniques include mathematical graphs, Markov chains, discrete event simulations, system dynamics, and agent-based modeling [158].

Finally, dealing with SoS engineering as opposed to classical systems engineering can result in the generation of a very high number of data points and gigabytes of information. Making sense of this data can reveal challenging and visual analytics tools may be used as one of the last steps of the SoS engineering process to handle this type of datasets. The purpose of such a step is to facilitate analytical reasoning for the decision-makers and provide deep insights by using a broad spectrum of visual representations. Analysts can then observe the data in multiple ways and also interact with it.

This section has drawn a parallel between the design of swarming systems and the field of systems of systems engineering. They share the same technical requirements of optimizing individual systems in order for them to fit in a bigger system. The challenges faced in the design of swarm robotics systems are similar to the ones encountered in SoS engineering. Hence the techniques of SoS engineering can possibly apply to swarming systems. An architecture-based design process was presented and justifies how some of its tools are relevant for swarm engineering, in particular modeling and simulation. However, such methods are not easily automated and tend to put more emphasis on managerial and operational factors than swarm engineering does. Keeping the methodology of SoS engineering in mind, swarm engineering also proposes a variety of swarm design methods which are presented in the next section.

2.2.2.1.2 Behavior-based design

As the most common design method, behavior-based design consists in an iterative process between the microscopic level and the macroscopic level of the swarm. The individual behavior of each robot is implemented and improved until the desired collective behavior is obtained. This type of approach enables to base the design on the behaviors observed in animal swarms which can ease the design process since mathematical models may already be available. This design method is based on trial and error and is typically a bottom-up process. Several techniques of behavior-based design are introduced in this subsection.

Probabilistic finite state machine design: during its mission, a robot uses the history of its sensors inputs to support its decision process and change its state. Such behavior can be modeled with Probabilistic Finite State Machines (PFSMs) where the transition between each state (see Figure 2.14 (a)) occurs based on a probability which can be fixed or change over time.

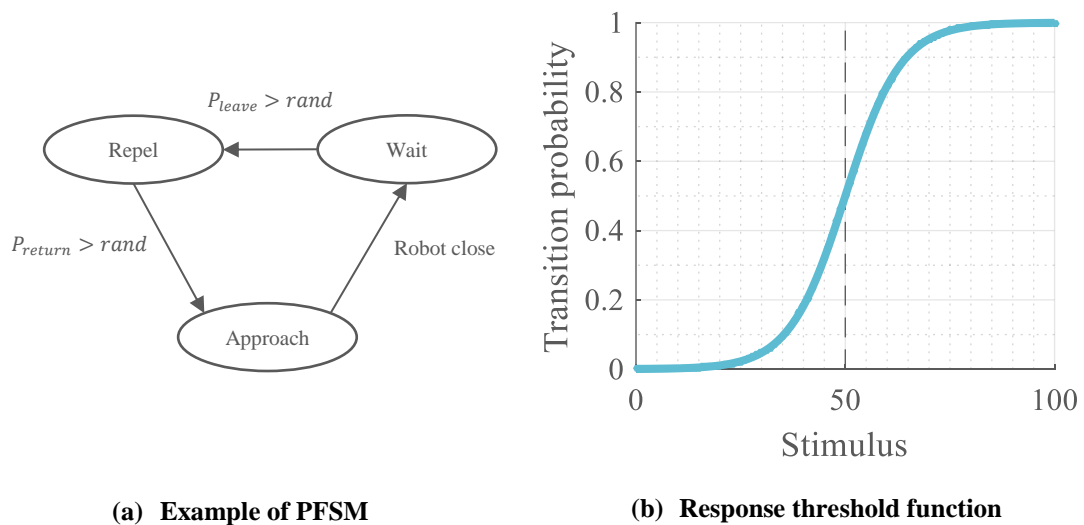


Figure 2.14: Probabilistic finite state machine design

For instance, [161] uses a fixed probability to implement an aggregation behavior in which robots decide to move towards or away from other robots based on a fixed probability. For time-varying or more generally non-constant probability, a mathematical function depending on the parameters of the system is used such as the common response threshold function (Figure 2.14 (b)) which is widely used for decision-making and task allocation [95].

Virtual physics-based design: in this design method inspired by physics, each robot is considered as a virtual particle exerting virtual forces on its counterparts. Individuals of the swarm are also subjected to virtual forces emanating from the environment. The goals of the mission are associated with attractive forces while obstacles may be modeled with repulsive forces. For instance in the physicomimetics framework, a robot computes a virtual force vector as $f = \sum_{i=1}^n f_i(d_i)e^{j\theta_i}$ where θ_i and d_i are the azimuth and distance of the i -th perceived obstacle or robot. The function f_i derives from an artificial potential function such as the Lennard-Jones potential (Figure 2.15) where the potential increases close to the obstacle or robot.

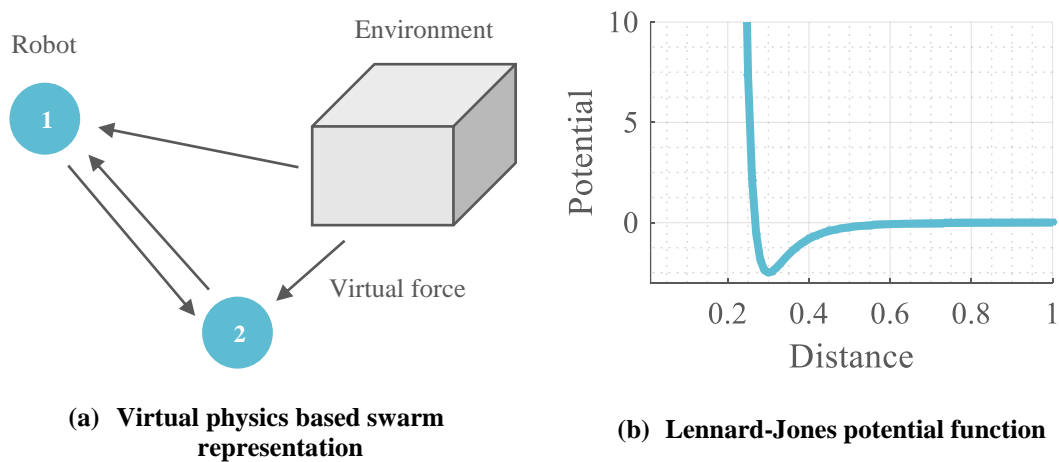


Figure 2.15: Virtual physics-based design

The stability and robustness of such methods can be proved theoretically and vector quantities are easily added to obtain global behaviors. Moreover, a single mathematical expression is required to account for the sensors inputs and translate these into actions.

Other design methods: other behavior-based methods are also presented by [95] such as the Protoswarm scripting language enabling the automatic generation of individual behavior scripts from a collective behavior. Another method focusing on the design of the swarm as a whole – and not only on the behavior design, is the top-down method from [162]. In this method swarm properties are first defined before the implementation of a macroscopic model. A model checker then verifies the properties in the model before a simulation implementation. Finally, the whole system is tested on real robots. This methodology helps verifying formally that the final system satisfies the established requirements, but remains a tedious iterative method based on trial and error.

2.2.2.1.3 Automatic design

Automatic design methods were created in the hope of reducing the effort of developers when trying to create collective behaviors. They are further classified into multi-robot reinforcement learning and evolutionary robotics.

Multi-robot reinforcement learning: robots can use reinforcement learning methods to learn a behavior through trial and error based on interactions with the environment. Positive or negative feedback is received by the robot based on its actions, encouraging a certain type of them and finally converging into an overall optimal policy. Optimality is considered here with respect to the policy that provides maximum reward from the interactions with

the environment. Reinforcement learning hence enables the generation of behaviors without an explicit implementation from the developer.

Using this principle with multi-robot systems, the difficulty lies mainly into dividing the global reward for the swarm into individual rewards for its agents: this is known as the spatial credit assignment. In addition, other factors limit the feasibility of multi-robot reinforcement learning. First, the considered state space is enormous since at a given step of the learning process, the possible actions are virtually infinite. Indeed, all possible decisions at the macroscopic level should be considered as well as all possible actions from each of its robots, making the number of possibilities very high. Moreover, the perception of the environment is imperfect and this latter is non-stationary as it evolves due to the actions of the robots.

Evolutionary robotics: as another concept inspired by nature, evolutionary robotics uses the Darwinian principle of natural selection and evolution in order to determine an optimal individual behavior to implement. This method refers to the steps of a classical genetic algorithm with individual behaviors as the individuals of the population. A random population is first generated and evolves through cross-over and mutation until the fitness of the population converges. The best individual of this population can be considered as the best individual behavior. To evaluate the fitness of an individual of the population, the mission of the swarm is executed with the given individual behavior (population individual). Note that this particular design method is particularly suited for homogeneous swarms since an individual of the population is only one individual behavior. In this design

methodology, an individual behavior can for instance be represented as a finite state machine as presented earlier.

Several other design methods that do not fit in any of the two previous categories are presented by [95]. These methods present the same characteristic as multi-robot reinforcement learning and evolutionary robotics as they focus on the behavior of the swarm and not its physical design. For example, [163] proposed a multi-robot architecture named L-ALLIANCE focused on task allocation in teams of robots.

2.2.2.2 Analysis methods

For the analysis phase, a swarm engineer focuses on checking in detail whether a given design produces the expected behavior or not. As it was mentioned in the previous sections, a swarm is modeled based on two different levels: the microscopic level which represents the single individuals of the swarm, and the macroscopic level modeling the characteristics of the entire group. Models which analyze swarming systems using both levels of interaction are still under research, especially the link between the microscopic level and the macroscopic level. Subsequently, the largest part of modeling techniques focuses on one level at a time [95]. An additional analysis technique consists in checking the implementation on real robots directly.

Microscopic models: they model each robot individually, accounting for its interactions with the environment, but also with other robots. Microscopic models can have a lot of detail in their implementation and vary from simple point masses, 2D representations, to complete 3D models with physics and intricate simulations of sensors and actuators. A

microscopic model explicitly embodies the behavior of each individual robot and is mostly analyzed through simulation. A benchmark to study scalability of such models was proposed by [164] using the Stage simulator. It showed that for this particular case, real time cannot be achieved once the population size exceeds a hundred entities in the most constraining cases. While some noise models are available as plugins, Stage ignores sensor noise. The simulator ARGoS presented in [165] is a modular and efficient simulator based on the segregation of space into subspaces, each running on different physics engines. The simulators studied in Table 2.3 can also be included in the microscopic models category depending on the detail level used in their implementation.

Macroscopic models: they focus on the swarm as a whole at a high level and do not consider each agent individually. A first approach to do so is to use rate and differential equations to model proportions of robots in the swarm and their state at a given point in time. However, modeling space and time dependency is difficult with such a method and modeling the communication between the robots can also be problematical.

Another method of modeling the macroscopic level of a swarm is by using the control and stability theory. Such techniques are used mainly in order to demonstrate properties of the swarm such as its stability or the existence of a global behavior. For instance, tools used to reveal such attributes are the Lyapunov stability theory, linear discrete-time dynamical systems, or delay differential equations. These methods are based on formal mathematical formulations but require many assumptions, not always satisfied in robotic swarms.

Additional methods are described by [95] but once again focus on the high-level behaviors of the swarm and not on its physical properties. For instance, [166] proved the convergence of a social foraging behavior in the presence of noise thanks to Lyapunov stability theory.

Mesosopic models: closely linked to simulation, extensive microscopic models and especially the 3D models, might contain too much detail for the purpose of conceptual design as they model every sensor and actuator. Coupled with the large design space exploration required for design optimization, this makes the use of detailed simulations quite long. However, an advantage of microscopic models is that the level of detail included can be varied and hence, adapted to conceptual design purposes. In particular, a fine balance between microscopic and macroscopic level has to be used: a sort of mesoscopic level modeling used to fill the gap between the aggregate level approach of macroscopic models and the intricate detailing of microscopic techniques.

Etymologically, the prevocalic “mes” means middle or intermediate and mesoscopic is usually defined as a scale between microscopic and macroscopic. The term mesoscopic is also found in applications other than modeling, as with the field of condensed matter physics for materials which length is between a quantity of atoms and materials measuring a few micrometers. Mesoscopic is also used in meteorology where mesoscale refers to weather systems smaller than cyclonic systems but larger than storm-scale ones [167]. Indeed, the synoptic (macro) scale size is around 1,000 km, the cumulus (micro) scale is below 5 km of horizontal dimensions and hence, the mesoscopic scale is left in between for systems ranging from five to several hundred kilometers. The mesoscale

is even divided into smaller subclasses. In the field of fixed-wing design, the microscopic scale would correspond to a complete dynamic model of the aircraft possibly including subsystems, the macroscopic scale would correspond to first-principles energy equations or even an agent-based model studying the interactions between aircraft, and the mesoscopic scale could correspond to response surface equations as used in [160]. As a limited microscopic model, mesoscopic models can be thought of as surrogates of microscopic models.

Such mesoscopic simulation techniques are notably used in transportation traffic management [168] and in Individual, Organizational, and Societal (IOS) research [169] focused on modeling human behavior as social units. Mesoscopic methods balance the required level of detail in different ways, for instance they may consider individual agents modeling, but not their interactions. Additional techniques used in traffic management include communicating cells of cars, queue-server approaches, or congregation of cars into packets travelling the network thanks to a speed-density function [168], [170]. Mesoscopic models are mainly applied when a sufficient detail of microscopic simulation is desirable but infeasible due to the size of the network or when the available resources in coding and debugging are limited. An example application to the aerospace field was proposed by [171] to propose a convenient modeling technique for congested airports.

Real-robot analysis: due to the difficulty of simulating a system of systems and all the details involved, an implementation on actual robots in order to validate behaviors is critical. Real experiments with swarms of robots help assessing the robustness of the design methods with respect to the noise present in sensors and actuators – amplified and spread

by the number of robots. It also tests the sensibility of the design to the simplifying and sometimes very reducing assumptions used in the models described earlier. Some real-robot implementations are just proofs of concept while other are extensive experiments. In both cases, managing such a fleet of robots presumes some important laboratory facilities and capabilities in buying or building a large number of platforms. Such experiments consume a lot of resources which explains why more than half of the research in swarm robotics presents simulation results without real-robot implementation. Moreover, these experiments are carried out in controlled environments, far from the possible mission conditions envisaged for real-world applications.

Collective robotics systems such as swarms are hard to design and the emergent field of swarm engineering tries to tackle this challenging task. In addition to the SoS engineering procedure, specific design methods of the swarm engineering field were shortly presented in this section. Most of these methods created for that field actually tend to focus on the implementation of behaviors with little or no focus on the physical design and constitution of the swarm. Moreover, these methods mostly concentrate on going from the macroscopic level to the microscopic level in order to derive the individual behavior of the agents based on high-level requirements for the swarm. Finally, the analysis methods try to include specificities from different levels but are still largely separated by level and seem to fail in clearly solving for the micro-macro link, with a possible exception for mesoscopic methods. However, these have never been used for swarm engineering. A summary of the most promising techniques studied in this section is proposed in Table 2.4

and Table 2.5 to help choose an approach answering the second research question. This comparison is based on the following criteria:

- **Individual agent:** quantifies whether the model is able to account for single agents individually.
- **Advanced individual capabilities:** assesses whether the model may include a high level of detailed modeling for the individual agents.
- **Swarm behavior:** measures if the method properly accounts for interactions resulting in swarm behavior.
- **Scalability:** quantifies the ability of the model to maintain its performance when the number of agents in the swarm is increased.
- **Noise modeling:** determines how easy it is to incorporate noise modeling in the method.
- **Can prove stability:** evaluates whether the proposed method can be used to derive the analytical stability of the swarm behavior.
- **Modularity:** rates how the model can be adapted to different types of swarming missions.
- **Implementation easiness:** quantify how easy it is to use or implement the proposed method. For instance, having to learn a new coding language results in a low score.

Table 2.4: Design methods review

Method	Advanced individual capabilities	Swarm behavior	Scalable	Noise modeling	Can prove stability	Modularity	Implementation easiness	Physical design
Behavior-based design methods								
Probabilistic finite state machine design	••	••	•••	•		•	••	•
Virtual physics-based design	•	•••	•••	••	•••	•	•••	•
Protoswarm	•••	••	•••	••		••	•	•
Brambilla et al. (2012)	••	•••	•	•••	••	••	•	•
Automatic design methods								
Reinforcement learning	••	•••	•	••	•	•	•	•
Evolutionary robotics		••	•••	••		••	••	•
ALLIANCE (Parker 1996)	••	••	•••	•		•••	•	•

Table 2.5: Analysis methods review

Method	Individual agent	Advanced individual capabilities	Swarm behavior	Scalability	Noise modeling	Can prove stability	Modularity	Implementation easiness
Microscopic level								
Point masses model	●●●		●●	●●●			●	●●●
2D models	●●●	●	●●	●●	●●●			●●●
3D models	●●●	●●●	●●●	●	●●●			●●
Vaughan (2008)	●●●	●●	●●●	●	●●●			●●
Pinciroli et al. (2012)	●●	●●	●●	●●●			●●	●
SwarmBot3D	●●●	●●●	●●●	●●●	●			
Mesoscopic level								
	●●●	●●	●●	●●	●●		●●	●●
Macroscopic level								
Rate and differential equations	●	●	●●●	●●●	●●	●●●		●
Classical control and stability	●	●	●●●	●●●	●	●●●		●
Liu and Passino (2004)	●	●	●●●	●●●	●●●	●●●		●
Real-robot analysis								
	●●●	●●●	●●●	●	●●			

From Table 2.4, design methods seem to provide good capabilities in terms of modeling a swarm behavior and most of them are quite scalable with the number of agents in the swarm. However, they do not account very well for the physical design of the agents since, as the previous paragraphs explained, they tend to primarily focus on behavior design [130]. On the other hand, analysis methods (Table 2.5) do consider the physical properties of the agents with a sufficient level of detail, especially microscopic models and real-robot analyses.

First, macroscopic models account very well for swarm behavior requirements as they try to follow a top-down approach. Often formulated as pure mathematical models, they are very scalable and may be used to derive certain analytical properties of the swarm such as its stability. However, such models are not the easiest to implement and they exhibit a lack of detail for the modeling of the individual agents.

Real-robot analysis is a very precise modeling technique since it performs experiments and missions with real robots to refine the design of the swarm. Although this method accounts pretty well for sensors noise and other experiment uncertainties, it remains impractical due to the time and cost commitment, especially in a conceptual design phase.

Besides, microscopic models represent the agents with great detail but due to the associated computational cost, they fail to scale efficiently with the number of agents. Moreover, the few microscopic methods able to scale up properly are designed for very specific types of missions and hence fail in the modularity category.

On the other hand, mesoscopic techniques are a tradeoff between modeling simplicity and adequate detail considerations. Such techniques usually describe the agents

at a relatively high level of detail, putting less emphasis on their behavior and their interactions which may be represented by macroscopic models. This type of detail balance is matching quite well with one of the intentions established earlier on: putting emphasis on the physical design of multi-robot systems at a conceptual design stage.

The previous observations based on a literature review of existing design techniques for group robotics indicate that moderate microscopic models, also known as mesoscopic models, seem the most adapted solution to solve the micro-macro link issue for conceptual design. Indeed, while macroscopic models have the ability to simulate very large groups of agents, they lack the level of detail required to account for the physical design of robots. On the other hand, microscopic models are able to precisely model the responses of individual agents but depend on many parameters, require extensive coding and calibration [172], and are limited in scalability owing to their computational cost. Originally used for transportation models and societal research, mesoscopic models fill the gap by providing modeling for individual agents while constraining the interactional behavior. They tend to have very strong capabilities for the modeling of both the agents and the swarm, and some of them also scale properly with the number of agents: key requirements to establish a micro-macro link in early design phases. This leads to the formulation of the following hypothesis:

Hypothesis 2

IF a mesoscopic approach leveraging the speed of macroscopic models and the accuracy of microscopic models is used
THEN microscopic and macroscopic levels can be efficiently linked for conceptual design purposes

The experiment formulated with the purpose of validating this hypothesis is described here below. Questions to be answered by this experiment include:

- How well does this apply to the conceptual design of multi-robot systems?
- In particular, is the approach fast enough for the exploration of a multi-architecture multi-level design space?
- How does this compare to other design techniques? (microscopic and macroscopic approaches)

Failure points for this experiment include cases when the simulation-based mesoscopic approach performs worse than microscopic or macroscopic methods from the literature. This is considered with respect to metrics relevant to conceptual design such as rapidity and precision. When compared with the mesoscopic approach, it is specifically expected that microscopic models perform much slower and with greater accuracy, and that macroscopic models perform much faster but with quite poorer accuracy. The accuracy metric is estimated based on whether or not the solution found by the approach is close enough to the performance of the real system. Note that random iteration cases can be tested on all methods to conclude on the rapidity of each one of them. However, for

complete insights on conceptual design, the whole optimization scheme has to be eventually considered. A set of validation criteria for experiment 2 is the following:

- The mesoscopic approach is not slower than current multi-robotics systems design space exploration methods
- The mesoscopic model of the tested mission is not slower than the microscopic model
- The achieved fidelity is sufficient for conceptual design purposes (20 to 25% validation error)
- The mesoscopic model of the tested mission is not less precise than the macroscopic model
- Ideally, the mesoscopic model speed and accuracy are the “average” of those of the microscopic and the macroscopic level

Required implementations for this experiment are:

- The testbed mission (see section 3.3.4)
- Mesoscopic model for a canonical mission: this model includes a detailed mission analysis for the microscopic level but simple group dynamics for the macroscopic level
- Microscopic and macroscopic models for the testbed mission for comparison purposes
- A swarm design space exploration technique, detailed in the following section

2.3 Exploring a large design space

As identified with the research challenges mentioned in the formulation of the research objective, the design of a group of robots quickly generates an extremely large design space (Figure 1.37 page 70). This design space is multi-architecture as several types of vehicles are considered, and also multi-level since not only there is a design space for swarm design variables at the macroscopic level (number of agents and control scheme for instance), but also for individual agents at the microscopic level (number of rotors and type of battery for example). A pertinent design space exploration technique must then be considered to account for this particularity and lead to an optimum swarm design. Before proceeding further on design space exploration techniques, the principal terms relevant with such a literature review must be clearly defined [173]:

- **Features:** functions or physical elements constituting a concept (for instance the wing)
- **Option:** a technical possibility in implementing a feature (for example a delta wing)
- **Alternative:** a given set of design variables which are sufficient to fully define a concept
- **Architecture:** a group of alternatives that can be described by the same design variables (multirotors for instance)
- **Configuration:** a given set of design variables that freezes the design of a given architecture (for example, a multirotor with 4 rotors, a battery of type LiPo and 700mAh capacity, arms of 10 cm long...)

As identified by [173], the literature distinguishes three main approaches to design space exploration: typical design process, architecture selection, and architecture configuration optimization.

The typical design process as mentioned by [173] is based on the infusion of technologies on existing baselines used as a reference for enhanced designs. These methods notably include top-down approaches such as Technology Impact Forecasting (TIF) [174], and bottom-up techniques similar to Technology Identification Evaluation Selection (TIES) [175]. These procedures try to reduce the overall risk and uncertainty by using very detailed baseline models selected by experts. However, this practice restrains the exploration of the design space to a limited local window around this baseline (Figure 2.16). The multitude of existing designs in the robotics field calls for broader design space explorations in terms of architectures selection without the limitations brought by a single baseline. Some exploration methods such as the one proposed by [176] require the designers to manually input the different architectures before optimization steps. Other architecture exploration methods such as [177], give advantage to a systematic generation and comparison of architectures without enabling architecture optimization. Inspired by the representations of [173], Figure 2.16 summarizes the capabilities of the three approaches to realize that no existing techniques enable for a dense enough exploration of the design space. Each patch represents an architecture and its size translates into the coverage of the design space proposed by the exploration technique and its different configurations.

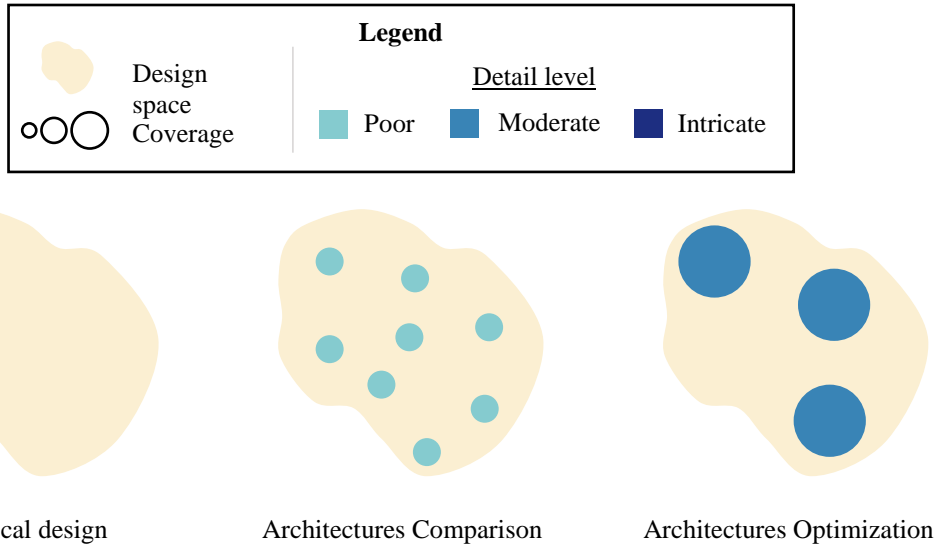


Figure 2.16: Capabilities of current design space exploration techniques

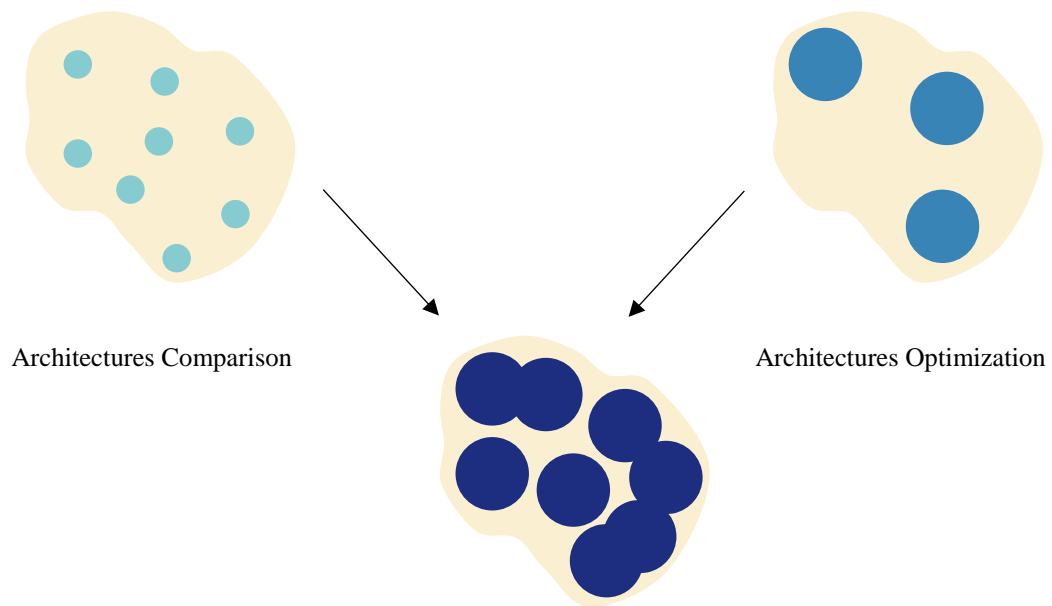
The typical design process offers a very detailed design space exploration but very limited in terms of coverage due to limited risk-taking. Architecture comparison methods study many architectures but at a poor level of detail, often dictated by qualitative assessments [173], and rarely considering more than one configuration. Finally, architectures optimization techniques offer a better coverage around a few baseline architectures at an intermediate detail level and with many configurations. Note that these observations hold true for the different levels of the considered design space.

This literature review shows that there is currently no adequate method for properly covering a multi-architecture multi-level design space for multi-robotics. This deficiency identified in existing approaches leads to a third research question:

Research question 3

How can current conceptual design methods be adapted to account for multi-architecture multi-level design space exploration?

As suggested by [173], a combination of architecture comparison and architecture optimization methods would potentially provide the ability to fully explore the whole design space with a proper coverage. Such a technique would be able to compare as many architectures as with the comparison approach while providing a decent coverage for each of these architecture thanks to the optimization step. Moreover, detailed physics-based models would enable for an accurate level of detail (Figure 2.17).



C. Frank proposed approach

Figure 2.17: [173] proposed approach

In particular, this method proposes solutions to some of the challenges mentioned here above and to the limitations of each of the existing approaches. First, the set of design variables describing the different alternatives of the “architectures comparison” method might not be the same for the different architectures. Moreover, each architecture might require its own optimization technique. Indeed, several baselines exist due to the possible heterogeneity of the multi-robot system and the design space is quite scattered, from well-known quadrotors to flapping wing designs. This prevents the use of a single optimization algorithm which would have to both optimize architectures and compare them at the macroscopic level. The architectures comparison method uses weak optimization processes which use a set of generic variables common to all considered architectures. These techniques are usually not able to precisely capture the different performance trends of each architecture. On the other hand, architecture optimization techniques focus on a subset of architectures which are described by the same design variables. Comparing these two philosophies, a tradeoff appears between the number of architectures considered and the achievable level of detail. The methodology proposed by [173] is able to systematically generate alternatives for a single-level design space, and also optimize alternatives described by different design variables. However, this method has to be adapted to the design space of swarming systems, characterized by multiple levels in addition to multiple architectures. Moreover, due to the extreme proportions of the design space, the accurate modeling and simulation environments recommended by [173] cannot be implemented for this work and mesoscopic models have to be considered instead. This proposed modified approach is illustrated on Figure 2.18.

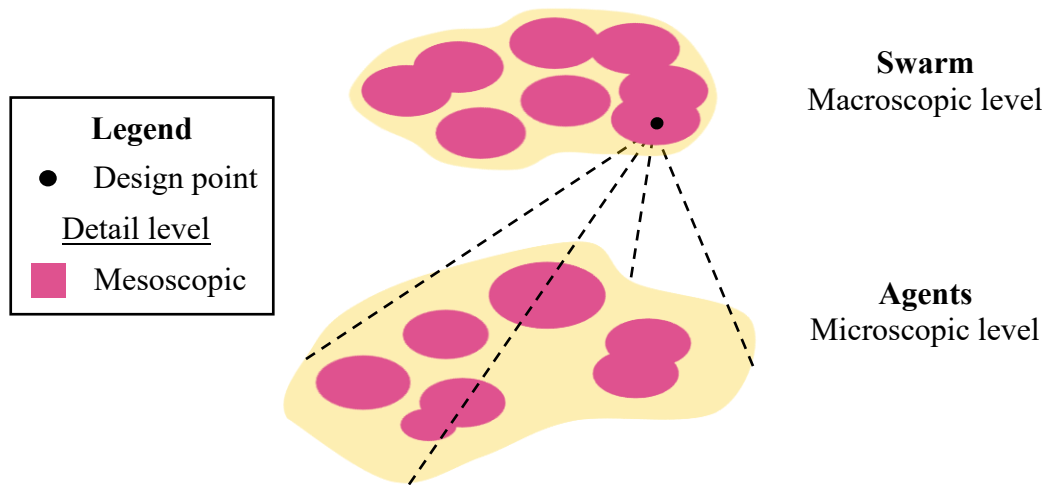


Figure 2.18: Proposed approach

After this brief explanation of the proposed approach philosophy, the third research question may be broken down into two sub research questions focusing first on the generation of alternatives and then on the optimization of the architectures, detailed in the next subsections.

2.3.1 Generating alternatives in a multi-architecture multi-level design space

This first step has to provide the ability to generate alternatives which might not have been studied yet and cannot consist in the sole enumeration of existing architectures. In addition, one has to keep in mind that the alternatives are generated in order to be later optimized. This optimization process might possibly be computationally expensive and as a consequence, the number of generated alternatives should ideally be minimized. As a consequence, full factorial or other large designs of experiments (DoEs) can be excluded. Moreover, during the generation of alternatives, the number of possibilities may be reduced

by considering compatibility between the different choices involved. For instance, a laser range finder sensor, usually quite heavy, might not be compatible with ornithopter designs which are typically tiny robots. This is well illustrated by the taxonomy of [178] shown on Figure 2.19.



HOBBYIST	MIDSIZE MILITARY & COMMERCIAL	LARGE MILITARY SPECIFIC	STEALTH COMBAT
Limited payload capacity	Moderate payload capacity	Larger payload capacity	Low observable features
Limited range/persistence	Moderate range/persistence	Long range/persistence	Low-probability-of-intercept/low-probability-of-detection data links
High-definition imagery/video transmission	Advanced radar	Low-probability-of-intercept radar	Higher resistance to adversary jamming
Autonomous GPS and waypoint navigation	Encrypted high-bandwidth data links	Enhanced jamming/electronic warfare	
	Limited jamming/electronic warfare	Beyond line-of-sight communications	
	Target identification and designation	Releasable missiles bombs	
	Communication relay function		

Figure 2.19: A capability-based taxonomy of UAVs [178]

From one category to another, the capabilities vary greatly and it is important to account for this type of classification. Indeed, for a homogeneous swarm imaging mission, one could imagine large vehicles covering a wide area at a limited resolution from a high altitude while smaller agents could cover pinpointed areas with an increased resolution at lower altitudes. A quantification of the performance of such categories is provided in Table 2.6.

Table 2.6: UAV capabilities by category

	Payload	Range	Real-time Data Transmission Range	Endurance
Hobbyist	Few kilograms	Few kilometers	2 km	15 min
Midsized Military & Commercial	100 kg	200 km	30 km	60 min to a few hours
Large Military-Specific	Over 1,000 kg	A few thousand kilometers	Global (300 to 800 km without satellites)	24 hours or more
Stealth Combat	Over 1,000 kg	A few thousand kilometers	Global (300 to 800 km without satellites)	5 to 24 hours

All the previously mentioned challenges give rise to the following sub research question:

Research question 3.1

How can we systematically generate all feasible alternatives in a multi-architecture and multi-level design space for further optimization?

To find response elements, existing methodologies of alternatives generation are reviewed and compared in the next section.

2.3.1.1 Review of existing methods

Alternative generation methods range from creative thinking and brainstorming techniques to more exhaustive and systematic approaches, from linear methods to intuitive methods. Being able to automate the generation of alternatives is a key requirement to satisfy the research objective and this section will then focus on linear creative thinking techniques which use existing information to generate new ideas. [179] segregates such methods into three groups. A group A where methods reorganize known information in different ways by listing, dividing, combining, or manipulating it to yield new entry points for solving problems. Example of such methods include false faces reversal, slice and dice attribute listing, cherry split fractionation, or again think bubbles mind mapping. Group B encompasses methods which are focused on categorization and are hence more systematic and may possibly be automated. Methods from group B notably include force-field analysis, morphological analysis, idea grids, diagramming, the phoenix method, and the future fruit method. Finally, group C favors breaking out of old and established patterns of thought in order to reach uncharted creative territory. It contains random stimulation, brute thinking, forced connection, pattern language, and the talk to a stranger method. Once again, the importance of automated alternatives generation motivates a focus on some of

the methods of group B. Common techniques are described here below and later leveraged to provide possible answers to research question 3.1.

Force-field analysis: also dubbed tug-of-war, this method was introduced in 1946 by [180] with main contributions to group dynamics and action research. It relies on the idea that a status quo, in design configuration for example, is held in balance by a set of forces. Some forces are driving and tend to promote change, while some other restraining forces attempt to maintain the status quo. Force-field diagrams (Figure 2.20) help the designer understand the “tug-of-war” between the forces at stake by representing the different forces and their importance.

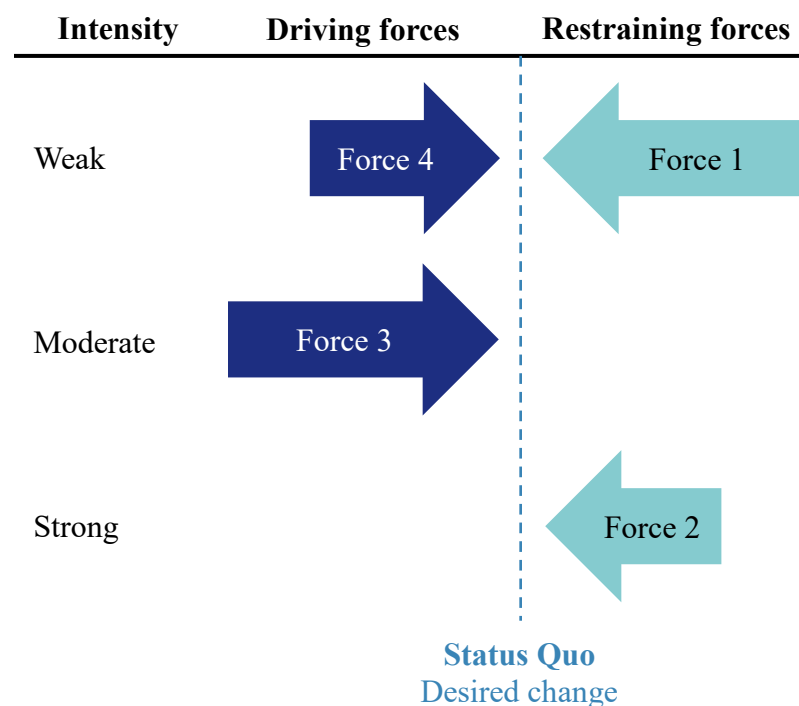


Figure 2.20: Force-field analysis

The designer then moves away from the status quo by acting on each of the forces enforcing the equilibrium. While this method is quite graphic, useful and easy to understand, it remains rather subjective. Moreover, modifying one force in the diagram might affect some of the other forces and this dependence is not accounted for in the original method.

Theory of Inventive Problem Solving (TRIZ): translated from Russian to “Theory of Inventive Problem Solving”, the TRIZ method is based on an extensive review of invention patents and the identification of patterns in that study. The approach relies on 40 principles and 76 standard solutions identified from the patterns and which may be applied to a status quo in order to obtain non-compromise solutions. In particular, a contradiction matrix is created to categorize conflictual elements and solve the issues thanks to successful past implementations. Many variations of the TRIZ techniques exist and focus on different stages of the process. Nevertheless, the TRIZ technique requires an important amount of knowledge and data to be able to perform the patterns identification. Moreover [181] underlines that this method is mainly human-oriented and challenging to implement on a computer.

Morphological matrix: based on functional and physical decompositions of a system, the morphological analysis lists all possible alternatives for a given function or feature [182]. In its matrix form, the rows represent the features while the columns represent the options for that feature. This approach is very easy to automate on a computer given that a database of features and options is available. However, as a full factorial approach, it tends to

generate a very high number of alternatives with a portion of incompatible ones. For instance when designing a multicopter, an autopilot motherboard might not accommodate a given number of rotors. To tackle this issue, the morphological matrix is often coupled with compatibility matrices for each pair of features indicating whether an option is compatible with another. If option i of one feature is compatible with option j for the other feature, then element (i, j) of the compatibility matrix is 1 and 0 otherwise. Considering a system of n functions, a total of 2^n compatibility matrices is required. While this requires additional work, the compatibility approach greatly reduces the number of total alternatives and it has been implemented in many software suites.

Decision tree: this approach uses tree graphs to map all possible paths to reach an alternative. Each node of the tree is a decision step between several options and compatibility issues are already addressed at this level [183]. Decision trees are simple to interpret and very flexible when adding new design options. Still, they require the designers to consider all branching possibilities manually. Moreover, their implementation in terms of data structures and memory management remains slightly harder than for a morphological matrix.

Exploration for emergence: at the frontier between architectures generation and optimization, [140] proposes a method to explore the design space by focusing on the emergent behavior of swarming systems. Divergence measures are used with adaptive sampling methods in order to yield the greatest amount of knowledge about emergence. However, this approach is limited to pure swarming systems as defined by [95] and may

not be suitable for the more generic intentional cooperative robotics problems introduced earlier.

Interactive Reconfigurable Matrix of Alternatives (IRMA): proposed as a collaborative design tool, IRMA enables to incorporate tacit information into the concept selection process [184]. Compatibility matrices and filtering options are used to downselect subsets of architectures. The main advantage of the IRMA approach is to provide a traceable reduction of an astronomically large design space to a manageable set of alternatives. However, this method requires a thorough understanding of the technologies at stake in order to populate the different filters. Moreover, IRMA is not very suitable for multi-level design spaces without additional modifications. M-IRMA tries to tackle this issue by handling different mapping levels based on a functional decomposition [185]. A qualitative or quantitative measure of performance is performed on the alternatives based on low-fidelity models, high-fidelity models, or physical experiments. While this enables a drastic down-selection of the configurations not meeting requirements, it also critically slows down the process. This is a problem for extremely large design spaces such as the one considered in the scope of this research. Moreover, the filtering step is based on an estimated performance of the system, which might not be accurate so early in the design process. This might possibly eliminate designs which could actually be promising.

A summary of the different methods studied in this section is proposed in Table 2.7 with their different advantages and limitations.

Table 2.7: Review of alternatives generation methods

Method	Number of alternatives	Can be easily automated	Objectivity	Accounts for compatibility	Dynamic	Suitable for mesoscopic
Force-field analysis [180]	••	•	•			
TRIZ [181]	••	•	••	••		
Morphological matrix [182]	•	•••	•••	•••		
Decision tree [183]	•	•	•	•••	•	
IRMA [184]	•••	•••	••	•••		
M-IRMA [185]	•••	••	••	•••	••	

After reviewing the existing methods for generating design alternatives, the decision tree seems quite interesting as it enables to easily add new alternatives and could hence be a good candidate to implement a dynamic generation of alternatives which would depend on the composition of the swarm. In addition, the morphological approach seems appropriate and rigorous to consider a sufficient number of alternatives by decomposing the system into features. As a matter of fact, one of its most common form is the morphological matrix, used in numerous aerospace design methodologies [186], [187]. Moreover, [188] introduced the notion that the morphological analysis approach can be used beyond the physical representation of individual systems and would hence be able to represent the whole system even if this method has to be adapted to account for a dynamic design space. This concept is introduced as the Augmented Morphological Matrix (AMM) [188]. The morphological approach is extensive and may generate an extremely high number of alternatives, a factor amplified by the need for iterations for the further optimization algorithms. Hence, this number of alternatives needs to be reduced. Moreover, the morphological approach does not account for the fact that the architectures should be described by the same sets of design variables in order to be later optimized. Indeed, after the alternatives generation process, several architectures with different design variables might be given to the optimization algorithm but the set of variables to optimize for a quadrotor is different than from an ornithopter design. Disabling some design variables based on the architecture is not a viable option: these design variables might be considered as good values by the optimization scheme and hence bias the optimum result for other architectures where these silent design variables would not be needed. The morphological approach has to be modified to account for this couple of challenges.

The method proposed by [173] is a first part of a solution to this research question as it considers a multi-architecture morphological approach to design space exploration. It first highlights that although a sequential use of conventional morphological and compatibility matrices would enable a systematic generation of alternatives, it is incompatible with further comparison and optimization of architectures [173]. A two-step process is then proposed to address this issue. The first step consists in grouping options which can be described by the same set of design variables. The main effect is to reduce the number of options available for each of the features, thus reducing the total number of architectures. The design variables which are used to describe the options of a given group are then directly included in the optimization process. As a consequence, the number of architectures is artificially reduced while the number of possible alternatives is increased [173]. The second step of the process removes the features that are described by only one group of options. As a result, the feature can be accounted for directly in the optimization algorithm via the related design variables. A last step consists in computing the conventional compatibility matrix out of this enhanced morphological matrix. An example of this methodology is applied to a notional fixed-wing UAV morphological matrix in Table 2.8 in order to obtain Table 2.9.

Table 2.8: Notional UAV morphological matrix

Features	Options			
Launch	Self-propelled	Catapulted	Hand-launched	
Landing	Conventional	Gliding	Parachute	
Wing	Delta	Swept wing	Straight wing	None
Vertical surface	Vertical stabilizer	Wing tip	None	
Jet engine	Typical turbofan	Augmented turbofan	Typical turbojet	Augmented turbojet
432 alternatives				

Looking at Table 2.8, all considered wings may be described by the same design variables such as a sweep angle, a surface area and an aspect ratio. Hence, the three types of wing are grouped and these corresponding design variables are integrated in the optimization algorithm. As for the jet engine options, they can also all be described by a unique set of design variables, triggering the removal of this feature from the morphological matrix. The design variables will be directly considered in the optimization process. These modifications being applied, Table 2.9 is obtained.

Table 2.9: Enhanced morphological matrix

Features	Options		
Launch	Self-propelled	Catapulted	Hand-launched
Landing	Conventional	Gliding	Parachute
Wing	Yes	No	
Vertical surface	Vertical stabilizer	Wing tip	None
54 alternatives			

Note that on this particular example, the number of discrete architectures is divided by 8 and this number grows exponentially with the size of the morphological matrix. This approach helps in capturing all alternatives while decreasing the number of executions of the optimization algorithm when compared to the approach of the sequential morphological matrix. However, while this particular approach enables to accommodate multiple architectures for design space exploration, it cannot directly be used for dynamic multi-level design spaces where architectures combinations depend on the alternative chosen for the upper level (swarm level). The next section constructs a hypothesis by leveraging these design space exploration techniques into a novel approach.

2.3.1.2 Hypothesis

The proposed approach takes into account the fact that the design space is dynamic and its size depends on the macroscopic level alternatives. Indeed, if at the macroscopic level a set of four drones of architecture quadrirotor is chosen, the design space has to expand to accommodate the possible design choices for the four agents. If a set of three ornithopters and two trirotors is then chosen at the macroscopic level, the size of the design space will be different. The steps of the proposed approach are described here below.

Step 1: perform the morphological matrix reduction

Using the steps introduced by [173] and described earlier, the total number of alternatives for each architecture is reduced to prepare for the later optimization procedure. This step is based on the set of morphological matrices available for each type of architecture. For example, due to the different control schemes involved in their conception and other particular features of each of these types, trirotors, quadrirotors, hexarotors, and

other octorotors might be better represented with different morphological matrices. However, at a conceptual design point of view and with design space exploration purposes in mind, these architectures can all be regrouped under one multirotor architecture in the same fashion as the example of Table 2.8 and Table 2.9. As per the arguments of [173], they can be regrouped under the same design variables such as the number of rotors, the length of the arms, or the size of the central plate (Figure 2.21). Note that in addition, groups of options represented by the same design variables are also regrouped.

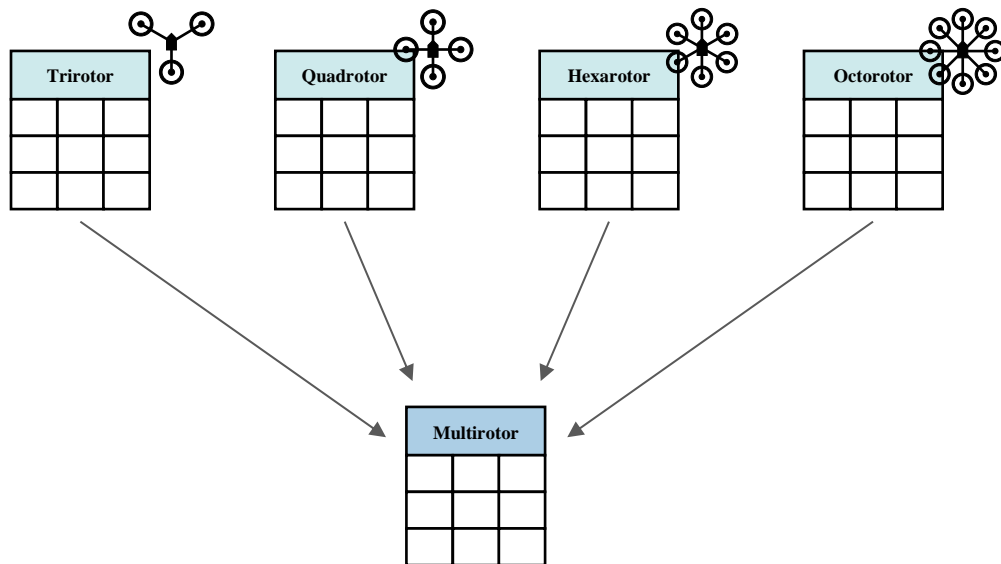


Figure 2.21: Morphological matrix reduction

While this concept is applied here to the simple and predictable case of multirotors, it is essential for the design space reduction step and the exploration of architectures in general.

Step 2: build the tree of morphological matrices

In order to be able to account for the evolving size of the design space based on the design choices made at the macroscopic level, a tree structure is implemented to keep track of the different architecture composition configurations of the swarm. The root of the tree is the macroscopic level morphological matrix while the leaves are the morphological matrices for each of the constituting agents of the swarm. At the intermediary level are conceptual or abstract morphological matrices used as templates for the leaves (Figure 2.22). Inspired by the decision tree approach described in the previous subsection, this proposed method enables to have a dynamic morphological analysis of a swarming system for generating alternatives.

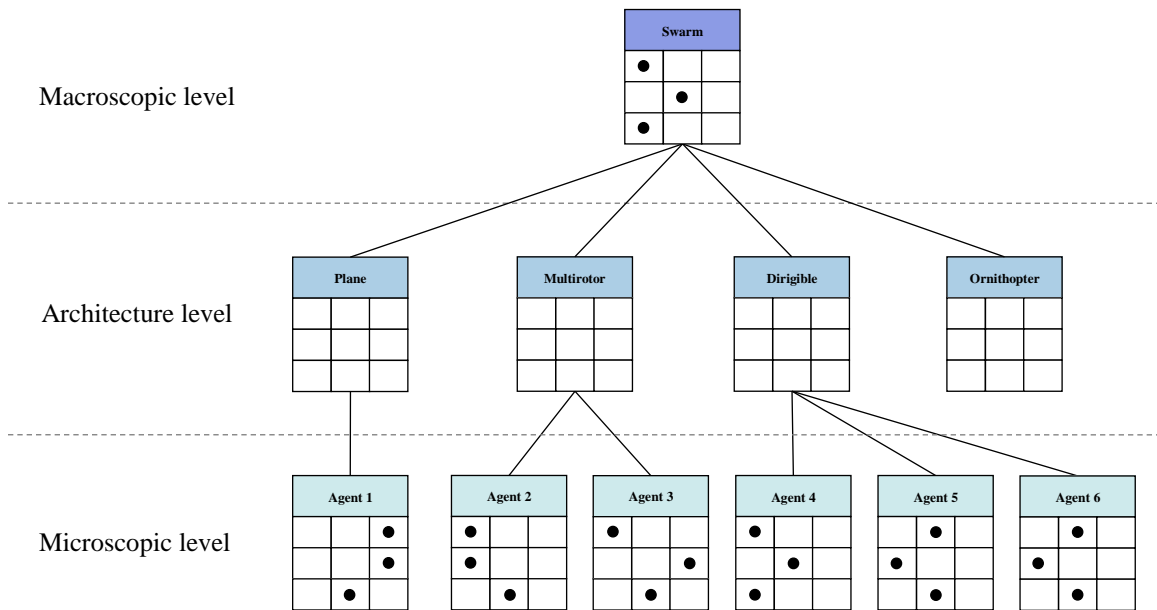


Figure 2.22: Example of morphological matrix tree

For instance by considering Figure 2.22, the swarm level dictates how many instances of plane or multirotor architectures are to be included in the design space. The

associated morphological matrices are generated as leaves under the corresponding architecture. The tree also handles the case when there is only one instance as in the case of the dirigible, and when there is no instance as for the ornithopter.

The whole proposed approach may be summarized by Figure 2.23 where both the morphological reduction and tree representation steps are represented.

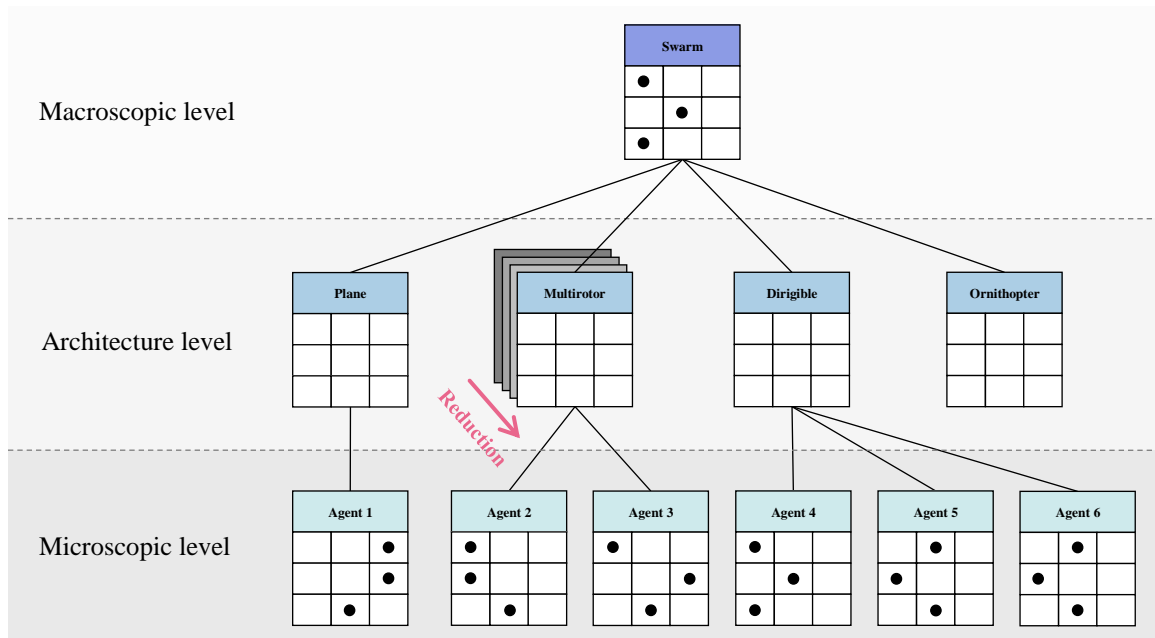


Figure 2.23: Proposed alternatives generation method

The elaboration of this technique then enables the formulation of the following hypothesis:

Hypothesis 3.1

IF a tree of reduced morphological matrices is used
THEN all feasible alternatives can be generated in a
multi-architecture and multi-level design space for further
comparison and optimization

The experiment implemented to validate this hypothesis first has to carry out a literature search in order to establish a representative list of possible features and options for existing architectures of UAVs. A set of morphological matrices is then created and reduced using step 1. The conforming compatibility matrices also have to be formed. The resulting new set of morphological matrices is then used to implement the tree structure required by step 2. Once these requirements are ready, alternatives are generated through the proposed methodology and the resulting coverage of the design space is evaluated with respect to relevant criteria such as: consistency, feasibility, exhaustiveness, as well as integration with respect to the whole methodology. Hence, validation criteria and questions to be answered by this experiment include:

- Are all alternatives feasible?
- Is each architecture defined by a unique set of variables?
- Are there still redundant variables or options groups in the resulting alternatives?
- Is the number of alternatives reduced when compared with the classical morphological approach?
- Are all existing concepts covered by the generated alternatives?

- Can the generated alternatives be easily fed to the optimization and analysis algorithms?

The next section details the second step required to generate alternatives: the optimization of the architectures.

2.3.2 Optimizing in a multi-architecture multi-level design space

A multi-architecture and multi-level design space present challenges for the optimization of generated alternatives against multiple criteria:

- The architectures considered to constitute the swarm are not always defined by the same design variables so that a single conventional optimization algorithm may not be used as it was established earlier.
- There are highly non-linear relationships present between the design variables of single agents and the group behavior, a multitude of design variables, and architectures. This prevents the use of surrogate models which could have sped up the design space exploration.
- The variables describing the alternatives maybe either discrete (number of motors) or continuous (geometric features). This prevents the use of the very fast gradient-based optimization algorithms.
- It is expected that many local optima exist since a multitude of robot combinations could lead to pseudo-optimal performance.
- Multiple objectives have to be optimized so that multi-objective optimization techniques have to be used.

- One slight modification at the vehicle level may completely disrupt the system level.

These challenges call for the development of an appropriate optimization process and the following research question:

Research question 3.2

How can swarm architectures be efficiently optimized
in a multi-architecture multi-level design space?

The next section presents a variety of methods found in the literature which could possibly be leveraged for the optimization of swarm architectures.

2.3.2.1 Review of existing methods

Despite a few attempts such as [189] or [190] to adapt gradient-based optimization methods to the issue of mixed variable types, such techniques remain better suited for problems involving only continuous variables and few local optima. Metaheuristic optimization approaches are thus preferred as they present several advantages related to the particularity of this work on robotic swarms:

- Stochastic algorithms are used, a key asset to find a global optimum.
- No gradient or Hessian information is required. This is appropriate here since given the lack of microscopic-macroscopic link, there is very little chance that expressions maybe obtained for analytical gradients. Moreover, finite differences methods would greatly handicap the execution time and lengthen the design space

exploration by several orders of magnitude which is not an option for conceptual design phases.

- Metaheuristic methods are often inspired by analogies with nature, physics or biology which reminds of the nature-inspired character of swarming systems.

In addition, it is important to notice that the problem of optimizing a swarm design has two main characteristics: multiple objectives are used in the optimization, and multiple levels (at least microscopic and macroscopic) are present in the optimization process. This motivates a review of the different methods which exist in both fields and could be used as response elements for research question 3.2.

2.3.2.1.1 *Multi-objective optimization*

Multi-objective optimization problems are usually formulated as shown on Equation 2.9 with x the vector of design variables, y the vector of cost objectives to be minimized with respect to x , and g and h respectively inequality and equality constraints.

Equation 2.9: Optimization problem formulation

$$\min_x y = f(x)$$

$$g(x) \leq 0$$

$$h(x) = 0$$

Approaches to solve this problem are usually split into two categories: a priori optimization and a posteriori optimization. These two philosophies are reviewed in detail in the next subsections to help the development of the new process.

2.3.2.1.1.1 A priori multi-objective optimization

As their name indicates, these techniques require a decision-maker to establish the relative preference of the different objectives prior to the search of an optimum. The extensive review proposed in [191] describes the following major approaches.

Lexicographic ordering: the decision-maker ranks the objectives in order of importance and the optimum solution is then obtained by optimizing the objective functions in sequence by starting from the most important one. This method tends to favor objectives one by one when in reality groups of objectives are usually considered. The optimal solution is hence performing well with respect to one objective over all the others, making it embarrassingly similar to a single-objective optimization technique. As a consequence, this method favors certain regions of the design space while neglecting some others. Lexicographic ordering is generally not privileged in specialized literature. One may also notice that this method is reminiscent of the sequential optimization gap identified in the current swarm design approaches.

Linear aggregating functions: the objectives are gathered in a linear combination fashion as in Equation 2.10. The weights w_i represent the relative importance of objective i estimated by the decision-maker. It is generally assumed for normalization purposes that $\sum_i w_i = 1$.

Equation 2.10: Linear aggregate function

$$f(x) = \sum_i w_i y_i(x)$$

Since only one optimization round is required, this method is generally faster than the lexicographic ordering approach.

Nonlinear aggregating functions: not commonplace in the literature, nonlinear aggregate functions use a barycentric approach of nonlinear functions of the objectives as shown in the example Equation 2.11.

Equation 2.11: Nonlinear aggregate function

$$f(x) = w_1 \cdot y_1^2 + \frac{w_2}{y_1 + y_2} + w_3 \sqrt{y_3}$$

Such approaches may be used when the importance of an objective is not static and may evolve based on its values. However, such methods usually require some additional preliminary work for the decision-maker to determine a set of conditions that these objectives functions must satisfy such as asymptotic behaviors or divergence issues. Moreover, the consistency of the aggregated objective functions is even harder to enforce in terms of units or physical meaning. Very often, defining a proper nonlinear aggregate objective function proves more difficult than the definition of a linear one.

Achievement scaling functions: also known as multi-criteria target vector optimization, this method is rarely used in the literature. The aggregate function accounts for the distance of the considered point to a desired solution [192]. A first simplistic implementation consists in using the Euclidean distance $d_E(t, y) = \sqrt{\sum_i (y_i(x) - t_i)^2}$ without accounting

for the difference in variance between the objectives. In that equation, $d_E(t, y)$ refers to the Euclidean distance between a target t and the current vector y of objectives. A more elaborated approach is based on a Mahalanobis-like distance $d_M(t, y) = [y(x) - t]^T \cdot S^{-1} \cdot [y(x) - t]$ where S is a matrix of weights, equivalent to estimated variances, for the different objectives. While this approach is generally used with a diagonal weight matrix, it is possible to account for an equivalent covariance between the different objectives by considering a non-diagonal matrix S . This approach is another way to represent the weighting scenarios of aggregate functions.

One of the difficulties in dealing with aggregate functions is the combination of the different objectives which might not be expressed with the same units. As a consequence, these quantities do not have the same orders of magnitude and this could lead to an objective involuntarily dominating the aggregate functions. This is the reason why the weights have to be carefully established and normalized. Two methods in particular enable to account for this issue independently of the weighting factors. The first one is a non-linear approach introduced by [193] and formulated as shown in Equation 2.12.

Equation 2.12

$$f(x) = \left(\sum_i \left[w_i \frac{(y_i(x) - y_i^*)}{y_i^-(x) - y_i^*} \right]^2 \right)^{\frac{1}{2}}$$

Where y_i^* and y_i^- respectively represents the target and the worst known value for objective i . The second one is especially applied to the aerospace domain and is called the

Overall Evaluation Criterion (OEC) [141], as seen in Equation 2.13. This method uses a linear aggregate function.

Equation 2.13: Overall evaluation criterion

$$OEC = \sum_i w_i \cdot \left[\frac{y_i(x)}{y_i^{BL}} \right]$$

In this formula, the reference used for achievement scaling is the value of the objective for a baseline (BL) concept.

Despite the limiting fact that these methods assume a prior knowledge of objective prioritization and might be highly subjective as such, they are easy to implement and require very little computation overhead. This is a non-negligible asset for the exploration of an exceedingly large design space. Moreover, they have the advantage of providing a unique optimum solution to the decision-maker. The subjectivity induced by the choice of the weights may be attenuated using replications of the process for different weighting scenarios. Finally, the aggregate function obtained from this process can be used as the objective function in classical single-objective optimization techniques which are more mature.

2.3.2.1.1.2 *A posteriori multi-objective optimization*

In a posteriori optimization, the multiple attributes of the y vector are not combined into an aggregate objective function. Instead, a set of solutions is generated, each of which represents a relative optimality between the competing attributes [194]. The designer then

has the choice of deciding of a “best” solution out of this set in a separate exercise of decision-making. The concept of a posteriori optimization is hence based on partial ordering and its goal is to find a set of designs which are better than all designs they can be compared to, but are incomparable to each other.

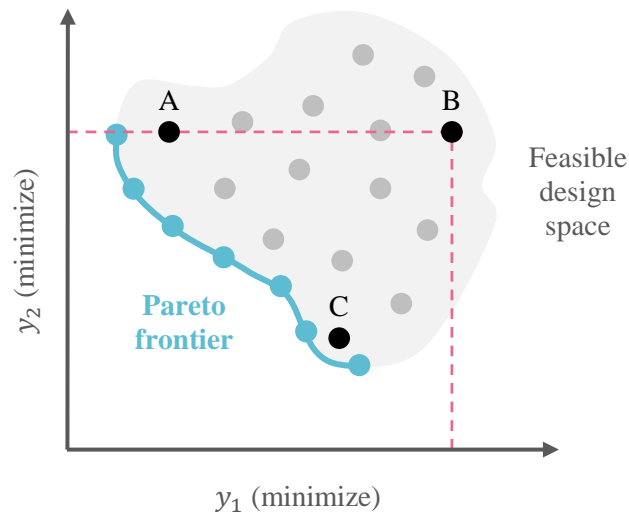


Figure 2.24: Partial ordering and Pareto frontier sampling

This set is referred to as the Pareto frontier (Figure 2.24) and has the following rules:

- A weakly dominates B if A is better in some attributes and equal in others
- C strongly dominates B if C is better in all attributes
- A and C are incomparable if A is better than C in some attributes but worse in others

While Pareto optimization does not provide a single optimized point, designers have the ability to represent the Pareto frontier and directly visualize and understand tradeoffs. This is particularly important when uncertainty is considered in the design approach. However, sampling the points of the Pareto frontier is a complex problem hard

to implement [173], [194]. This is amplified when the number of criteria, constraints, and variables considered is very large just as it is the case for swarming systems.

A first idea to sample the Pareto frontier is to solve a series of single-objective optimization problems, each one of them yielding a point of the Pareto frontier. Although it is possible to vary both the constraints and the objective function considered, common approaches usually define a weighted p-norm of the objectives (Equation 2.14):

Equation 2.14: Weighted p-norm of the objectives

$$f(x) = \left(\sum_i w_i y_i^p(x) \right)^{\frac{1}{p}}$$

Where y_i represents the objectives with their associated weights w_i in the overall objective function f . This basic principle is revisited with more elaborated methods such as the Epsilon constraint method, the Normal Boundary Intersection (NBI) method, and the Normalized Normal Constraint (NNC) method. While being fast and straightforward to implement, this set of methods exhibits deficiencies in their application to non-convex, highly non-linear and constrained design spaces [194].

An alternative approach to the sampling of the Pareto frontier is represented by evolutionary algorithms which optimize a population of points. As the algorithm iterates, the population of points converges to the true Pareto frontier. A few of these approaches are presented here below from the exhaustive review offered by [191].

Independent sampling techniques: efficient and simple, they are based on the linear aggregating function methods but the variation of the weights is not independent anymore and is directly included in the evolutionary process. However, these methods fail to produce an even sampling of the frontier.

Aggregation selection techniques (linear, nonlinear): slightly similar to the independent sampling techniques, these do not use a static weight combination throughout one run of the evolutionary algorithm but the weights are varied between generations and function evaluations. Different assignment schemes are usually used such as random assignment, gene-based assignment or fitness-based assignment.

Criterion selection techniques: well represented by the Vector Evaluated Genetic Algorithm (VEGA) method, such methods base the selection of a succeeding population on separate objective performance. This requires only a few changes from a classical genetic algorithm. However, the fitness evaluation of the individuals corresponds to a combination of the objectives meaning that VEGA is subject to the same limitations than the aggregating approaches previously discussed. Extensions of the VEGA method propose different schemes of weight assignment.

ϵ -constraint technique: a primary objective function is varied while others are bounded within allowable ϵ -constraints. These constraints are then adjusted to keep generating points on the Pareto frontier. While it remains fairly easy to implement, this method

requires a massive computational effort, is prone to non-uniformity of the sampled points, and the bounds of the constraints must be known a priori.

Pareto sampling techniques: these approaches are based on the fact that evolutionary algorithms are able to generate several points of the Pareto frontier in a single stochastic computational run. These methods are generally quite hard to implement.

Table 2.10 summarizes the advantages and limitations of these different multi-objective optimization techniques.

Table 2.10: Review of multi-objective optimization techniques

Technique	Subjectivity	Design space coverage	Scalability Speed	Implementation easiness
A priori				
Lexicographic ordering	●●●		●●	●
Linear aggregating functions	●●	●	●●●	●●●
Non-linear aggregating functions	●●	●	●●●	●●
Achievement scaling functions	●●●		●●●	●●●
A posteriori				
Independent sampling	●	●●	●●	●
Aggregation selection	●	●●	●	●
Criterion selection	●	●●	●	●●
ϵ -constraint	●	●●		●●
Pareto sampling	●	●●●	●	

Although they generally provide a better coverage of the Pareto frontier in a variety of cases not handled by a priori techniques, a posteriori techniques are limited in scalability [191] as they tend to be complex in implementation and require the solving of many optimization problems. Indeed, the goal of such methods is to perform a search which is as widespread as possible so as to generate as many elements of the Pareto optimal set as possible. Finally, progressive techniques are another possibility for multi-objective optimization but rely on an interactive process with the decision-maker. Such an approach may prove difficult and inefficient [191] and is not appropriate in the scope of this research since only methods which can be fully automated are favored. In the light of this review of multi-objective optimization techniques, a priori approaches are preferred. However, since swarm robotics is an emerging field, no baseline or benchmark performance exists to use achievement scaling functions. Hence, the simple linear aggregate function is chosen in order to facilitate a fast exploration of an extremely large design space. In order to improve the robustness of such an approach with respect to the subjective choice of the weights, different weighting scenarios can be attempted and compared. This still requires less function calls than a posteriori optimization techniques.

2.3.2.1.2 Multidisciplinary optimization

The techniques of multidisciplinary optimization are generally categorized based on their number of optimization levels: single-level, bi-level, and multi-level [195]. Hence, considering multidisciplinary optimization approaches is particularly appropriated for optimization in a multi-level and multi-architecture design space. Mature and commonplace methods are reviewed in this section to understand how they can be leveraged to help reach the research objective.

2.3.2.1.2.1 Singe-level techniques

Single-level multidisciplinary optimization approaches use only one optimizer at the system level. The analysis may be distributed to the different partitioned subsystems but the optimization is kept centralized at the system-level. The most common single-level approaches are described here below.

All-At-Once (AAO): all the variables of all disciplines are considered as optimization variables and the equations of each discipline are used as constraints. Thus, the designs are only consistent at convergence of the algorithm and there is no guarantee that at any iteration, the design will be feasible for all disciplines. If the algorithm experiences convergence issues and fails to reach a relative or absolute extremum, it will yield a design which is not only sub-optimal, but also not valid across the disciplines. However, this method has the advantage of not necessitating a complex analysis process.

Multi-Disciplinary Feasible (MDF): this approach includes an analyzer which, at every optimization iteration, solves the disciplinary equations using the design variables until additional coupling variables converge. This ensures that the solution is consistent across all disciplines at each step of the optimization process. However, this solution might be infeasible with respect to the constraints. A limitation of this method is that it requires a complex system solver which coordinates all the subsystems in order to return a consistent solution to the optimization algorithm. This is not only hard to implement but implies a significant computation time at execution.

Individual Disciplinary Feasible (IDF): the IDF method focuses on the discipline feasibility at each iteration rather than on the multidisciplinary feasibility. This latter is only achieved at convergence thanks to constraints added for each of the coupling variables. The IDF method has improved convergence properties when compared to MDF but moves the complexity of the analysis to the optimization step which still requires consequent computational resources. Moreover, if the optimizer fails to converge, the produced solution might be inconsistent. In general, IDF performs better than MDF when the coupling between the subsystems is significant.

2.3.2.1.2.2 Multi-level techniques

As opposed to single-level techniques, the multi-level optimization methods use multiple optimizers at the subsystem level in addition to the traditional optimizer at the system level. This type of approach is preferred when the scale of the problem is too large for a single optimizer to handle [195]. In these techniques, the analysis and the design are distributed amongst the different subsystems.

Concurrent Sub-Space Optimization (CSSO): the CSSO approach decouples the disciplines by letting each subspace carry out a separate optimization based uniquely on the design variables of that discipline. The coordination of all disciplines is handled by global sensitivity equations and a sensitivity analysis determining the non-local variables. This analysis can be carried out by equations or by response surface approximations in order to reduce the computational burden [196]. This method is useful for the industry as it is compatible with organizational features and the decoupling generally observed. However, the consistency of the final solution is generally mediocre due to difficulties in

coordinating the subspaces optimization processes. This also makes it hard to guarantee a robust convergence of the whole optimization process [197], [198].

Bi-Level Integrated System Synthesis (BLISS): the BLISS approach divides the optimization problem into an upper level and a lower level. The subsystems of the lower level optimize on their design variables while the common variables are considered as constants. On the other hand, the upper level uses the common variables for optimization while the local variables of the lower level are regarded as constants [199]. A gradient-based approach is then used to reach an optimum. Hence, the computational cost of the BLISS method is quite important and limits its scalability. One approach by [200] to tackle this issue is to use response surface methodology. However, this is not applicable to multi-robot systems due to the high non-linearity between microscopic variables and macroscopic responses.

Collaborative Optimization (CO): this method is especially focused on early design phases where all disciplines are usually considered on the same level. The optimizer at the system level establishes targets to be met by the partitioned subsystems and tries to minimize the system-level objective function. A set of equality constraints ensures that the design is driven towards consistency. At the subsystem level, the goal of the optimizers is to meet the targets and satisfy the constraints of the respective subsystems. With CO, each subsystem benefits from having its own optimizer, allowing for greater autonomy of the disciplines. However, the ability to handle coupling is limited since the interactions between the disciplines are handled by the main optimizer.

Analytical Target Cascading (ATC): this approach uses a cascade of optimizers to propagate the design targets from the top level to the lower levels of the hierarchy. These lower levels are optimized to meet the targets and the resulting responses are rebalanced to the higher levels in order to achieve consistency for the whole system. This iterative process is repeated until consistency is achieved globally for the targets and the responses. This approach is truly multilevel and considers analyzers and optimizers almost at the same level. On the other hand, it requires many executions and is computationally expensive.

2.3.2.1.2.3 *Summary*

A recapitulation of these MDO methods is presented in Table 2.11 in order to help decide on a suitable framework for the research objective. They are evaluated based on their convergence performance, the easiness of implementation, their scalability with respect to the size of the design space, and the incurred computational cost. Moreover, given the structure of the problem with a macroscopic level and a microscopic level, it is important to distinguish methods able to account for several levels, dynamic design spaces, and congregate the microscopic and the macroscopic levels. These last criteria help ensure that a method can handle several levels without necessarily separating them in the optimization process. This is essential in order to distance the proposed approach from the sequential optimization paradigm currently used by the research community.

Table 2.11: Comparison of MDO frameworks [149], [195], [199]

Method	Good convergence	Easy to implement	Scalability	Computational cost	Bi-level	Handles dynamic design space	Congregates micro/macro levels
AAO	•	•••	•••	•			
MDF	••	•	•	•••			
IDF	•	•••	•	•			
CSSO [197], [198]	•	••	••	••	•		
BLISS [199]	•	•••	•	•••	•••		
CO	••	•••	•	•••	••		
ATC	••	•	•	•••	•		

From Table 2.11, no method is able to directly handle the dynamic nature of the multi-architecture and multi-level design space. Moreover, no method considers a possible congregation of the different levels without modification. From this apparent gap, an adapted optimization method has to be designed, focus of the next subsection.

2.3.2.2 Hypothesis

As previously stated in this section, metaheuristic approaches are first preferred to establish an optimization method. In particular, genetic algorithms enable dealing with discrete, continuous, and categorical variables at the same time. However, they have to be adapted since several levels are to be handled at once and the architectures might have different design variables.

A first consideration in designing the optimization technique is that this latter has to contain at least two layers. Indeed, a single layer would not be able to handle the dynamic aspect of the design space provided by the macroscopic level design choices. However, it is important to notice that the segregation of the layers need not be done between macroscopic and microscopic levels. First of all, this would limit the congregation of the two levels and most probably yield optimization results similar to those obtained by the sequential optimization techniques. Secondly, the outer loop of the algorithm needs to take care only of the dynamic aspects of the design space which are to be later instantiated at the microscopic level. As such, only the types of architectures considered and their number in the swarm are susceptible to affect the size of the design space and the microscopic level implementations. Other design variables, even macroscopic, may be included in the inner loop optimizer.

In order to accelerate the process, it is possible to retain the good architectures, and possibly designs, from the inner loop so that they are used as an initial inner loop population for the next iterations of the outer loop. Indeed, since the mission is fixed during the iterations, it is probable that an architecture performing well on the mission in a given swarm might show good performance in another swarm configuration. Due to heterogeneity and highly unpredictable macroscopic level effects, it is not guaranteed that a good architecture for a given swarm configuration would also perform well in another. However, this assumption is susceptible to speed up the whole optimization process if the initial swarm populations are initialized with good designs.

To summarize the optimization scheme, an outer loop optimizes the types of architectures to include in the swarm as well as their number. This configuration is then fed to an inner loop which optimizes the multi-robot system based on the remaining design variables. A key particularity lies in the fact that the outer loop handles only macroscopic variables but the inner loop deals with both microscopic and macroscopic design variables. The advantages of this method are multifold:

- Microscopic and macroscopic levels optimizations are now combined, an approach different than the usual sequential optimization scheme.
- It provides augmented capabilities since it enables multi-architecture and multi-level optimization.
- The retention of optimal microscopic configurations accelerates the convergence process and the design space exploration.

- There is no need to implement a new optimization algorithm for each robot architecture, only two generic optimization algorithms are to be implemented.

Figure 2.25 illustrates both optimization loops with an implementation based on a genetic algorithm.

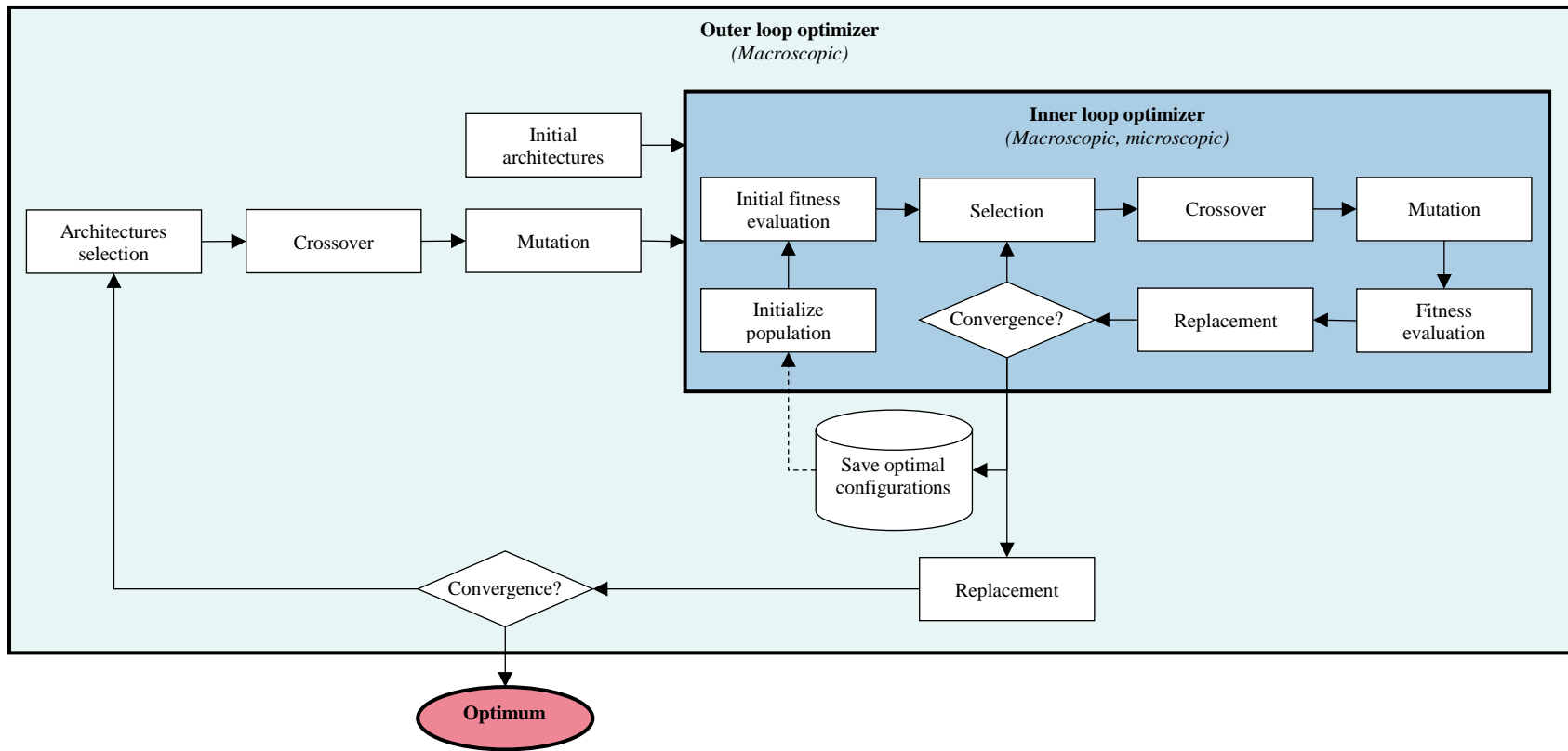


Figure 2.25: Proposed optimization scheme

The elaboration of this optimization scheme yields the following hypothesis:

Hypothesis 3.2

IF an optimization method based on a bi-level genetic algorithm is used
THEN a fast and efficient multi-architecture multi-level global
optimization of group configurations is enabled

The experiment designed to test this hypothesis is quite similar to experiment 1 and aims at assessing the quality of the optimization scheme. It first requires the implementation of both inner and outer loops, and a modular genetic algorithm. Then, the optimization algorithm is evaluated on the number of iterations, number of objective function calls, and precision in terms of optimal swarm performance. Again, additional metrics used to assess optimization algorithms can be used, such as the ones reviewed in [155]. To assess this precision, it is possible to compare the obtained optimum against a “ground truth” optimum obtained by randomized sampling, full factorial, or by taking the best known solution [156]. The parameters to be varied during this experiment 3.2 include the initial swarm population constitution, the typical genetic algorithm parameters, and the retention scheme for the optimized microscopic architectures.

Once again, the validation criteria of hypothesis 3.2 are based on statistical hypothesis evaluation:

- On average, the global optimization scheme is able to find a better solution than the sequential optimized solution, with respect to the main mission performance metric.

- The optimization scheme is fast enough for the design space exploration of multi-robot systems.
- The time increase between the proposed scheme and sequential optimization is less than an order of magnitude

2.4 Summary

After identifying the main steps involved in the realization of the research objective, this chapter concentrated on the stages requiring research advancements to fill existing gaps. A literature search was carried out on each of the necessary steps and deficiencies were identified between existing techniques and the goals of the research objective. Such inadequacies led to formal research questions. Additional reviews of the existing literature in relevant fields thenceforward facilitated putting together conceivable answers to the research questions. With the aim of testing and validating these hypotheses, a rigorous set of experiments was designed, hence closing the elaboration of the research process (Figure 2.26).

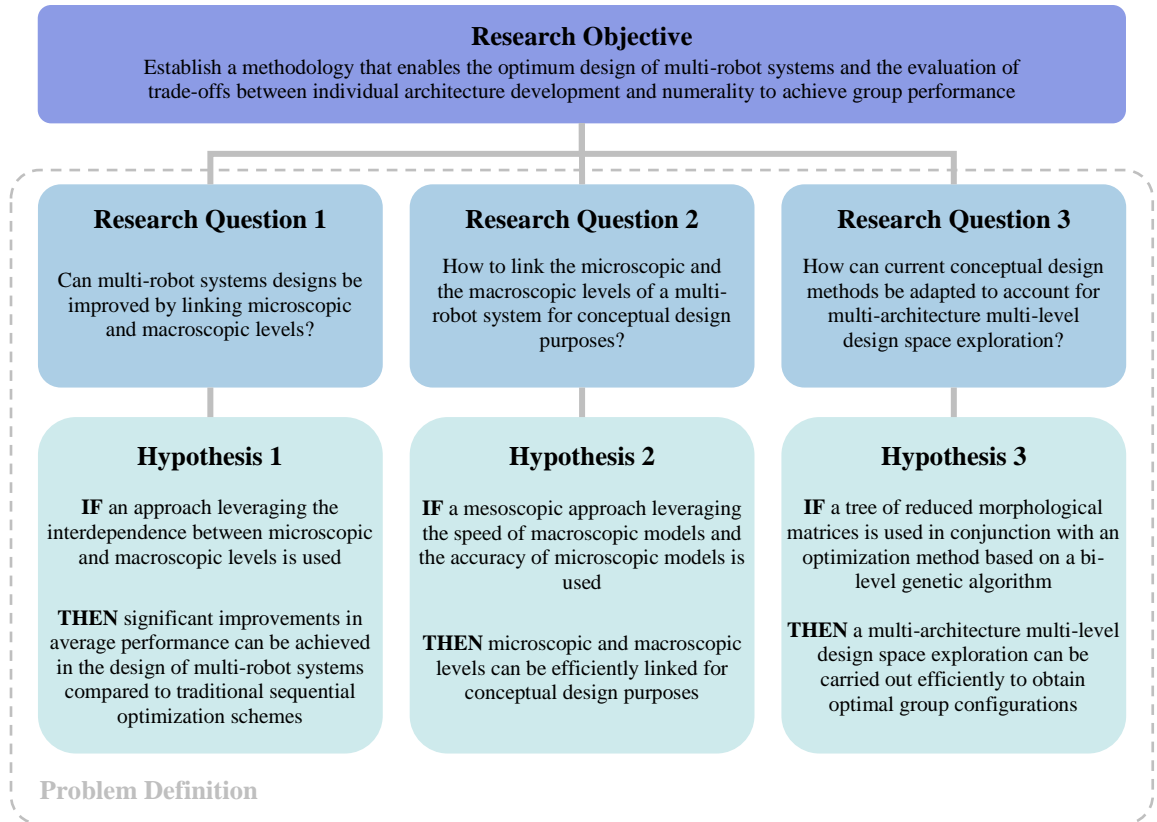


Figure 2.26: Summary of the problem definition process

The implementation details of the conceived experiments are discussed in the next chapter.

CHAPTER 3

PROPOSED APPROACH

Focusing on design considerations, the present work proposes to base its approach on a classical top-down design decision support process such as the one presented on Figure 3.1.

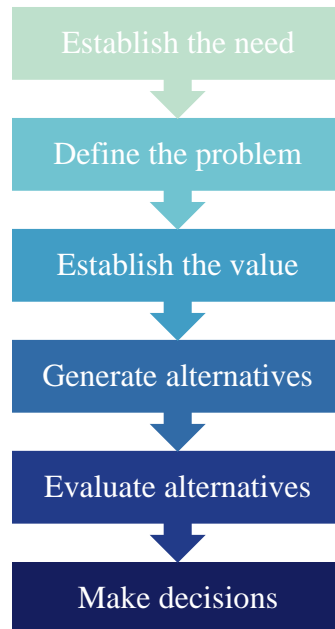


Figure 3.1: Generic top-down design decision support process

The first chapter presented different existing needs in the field of multi-robotics: take advantage of a rising diversity in the robotics fleet, evaluate the real advantage of multi-robot solutions, and elaborate a simultaneous physical design optimization of microscopic and macroscopic swarm levels. Based on these observations, a research problem was formulated to establish a methodology that enables the evaluation of dynamic

tradeoffs between individual architecture development and numerosity to achieve group performance in multi-robot systems. The second chapter then studied existing techniques to address the research objective but such procedures came up either inadequate or unsatisfactory after a thorough literature review. The chapter hence identified clear deficiencies in current practices which need to be overcome so as to meet the research objectives. These gaps originated the following main research questions:

1. Can multi-robot systems designs be improved by linking microscopic and macroscopic levels?
2. How to link the microscopic and the macroscopic levels of a multi-robot system for conceptual design purposes?
3. How can current conceptual design methods be adapted to account for multi-architecture multi-level design space exploration?

A second literature review supported the progressive formulation of hypotheses as possible answer elements to the research questions. These hypotheses are to be validated through experiments following the approach suggested in this chapter through a top-down design decision support process. Since the first two stages of this process were studied in the first two chapters, this section consists in detailing the remaining four steps: establishing evaluation criteria, defining the design space, evaluating design alternatives, and finally making decisions.

3.1 Establishment of performance metrics

A first step when studying different design alternatives is to institute performance metrics, evaluation criteria which are used to compare the different designs and optimize

them. Note that in this work, the sentences mentioning “good” performance or “efficient” result are a direct reference to these performance metrics. Such measures which may account for mission completion, swarm cost, or also parallelization effectiveness are presented in this section.

3.1.1 Parallelism efficiency metrics

While microscopic variables influence the performance of each individual agent, macroscopic variables impact the parallelization of the different tasks. In order to quantify the effect of such variables on a given design, parallelism efficiency metrics are required such as the parallelism efficiency and the Limit of Parallel Effectiveness (LOPE).

3.1.1.1 Parallelism efficiency

A common phenomenon experienced in parallel computing is parallel slowdown: parallelizing an algorithm past a certain limit results in the program to run slower. Ideally when running a computation on N processors, one should hope for a linear speedup of N times. However, the parallel implementation itself introduces various delays due in part to communication of intermediate results, cache misses, or resource contention [201]. Indeed, with an increasing number of processors, each parallel node of the algorithm spends more time in communication than in processing. Such delays occasion a slowdown compared to the ideal case and several related quantities introduced by the community are presented here below.

An intuitive way to quantify parallel slowdown is to simply compute the ratio of the time it takes a task to be executed in serial, to the time taken in parallel. Speedup is hence defined as:

Equation 3.1: Speedup formula

$$S = \frac{T(1)}{T(N)}$$

Where $T(i)$ represents the time for the task to be carried out using i processors. The actual parallel computation time $T(N)$ can then be compared to the ideal linear $\frac{T(1)}{N}$ case mentioned earlier, yielding the formula for parallelism efficiency:

Equation 3.2: Parallelism efficiency

$$\eta = \frac{\left(\frac{T(1)}{N}\right)}{T(N)} = \frac{S}{N}$$

Amdahl's Law is another attempt to study this slowdown behavior and quantifies how parallelization can speed up a computation. It segregates the tasks of a parallelizable program under two categories: a part which can be parallelized and a part which cannot. Let ρ the proportion of execution time which can be improved by parallelization, the total execution time can then be written as:

Equation 3.3: Parallelized execution time

$$T(1) = T = \underbrace{(1 - \rho)T}_{\text{Not parallel}} + \underbrace{\rho T}_{\text{Parallel}}$$

With parallelization of the tasks by N processors, and assuming a fixed workload for the processors, the proportion of tasks benefiting from it becomes $\frac{\rho T}{N}$ while the other

part remains unchanged. Rewriting the speedup formula (Equation 3.1), the common form of Amdahl's law is obtained:

Equation 3.4: Amdahl's law derivation

$$S = \frac{T(1)}{T(N)}$$
$$\Rightarrow S = \frac{T}{(1 - \rho)T + \frac{\rho T}{N}}$$
$$\Rightarrow S = \frac{1}{(1 - \rho) + \frac{\rho}{N}} \text{ for } T \neq 0$$

This equation can be refined further by using more detailed knowledge about the process to be parallelized and the decomposition of $T(N)$ into different tasks. The efficiency formula (Equation 3.2) can also be expressed by dividing the speedup by N :

Equation 3.5: Parallel efficiency with Amdahl's law

$$\eta = \frac{1}{\rho + (1 - \rho)N}$$

As more and more processors are added, the computation time becomes dominated by the non-parallelizable tasks. This places a limit on the speedup achievable by the system, causing the efficiency to tend towards zero (Figure 3.2).

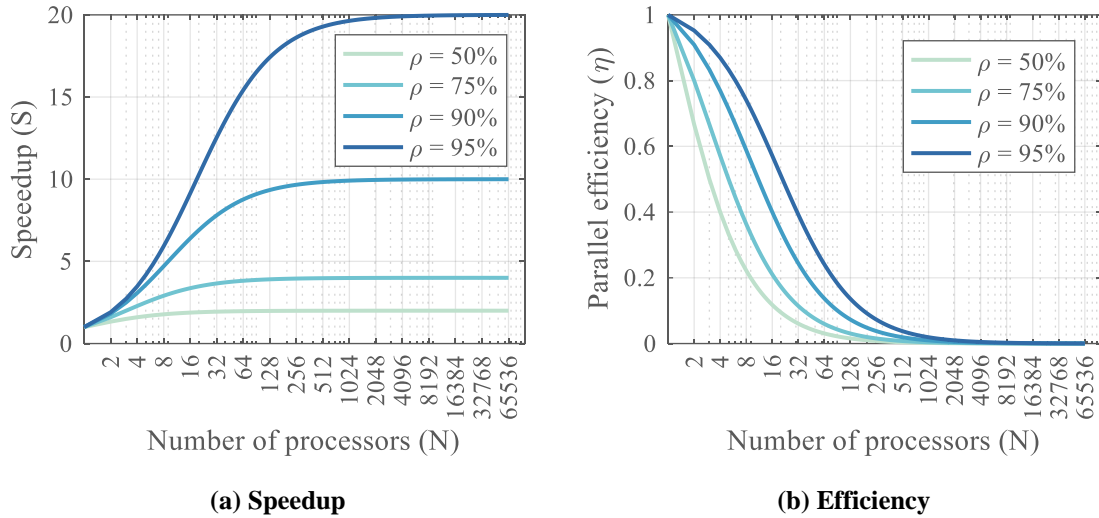


Figure 3.2: Amdahl's law

For example, if 95% of the process can be parallelized, the maximum speedup theoretically achievable by parallelization is 20 times. It can be noticed that the lower the proportion of parallelizable tasks, the lesser processors are required to reach the speedup limit.

Amdahl's law was later revised by Gustafson's law [202] by changing the assumption of a constant workload, equivalent to a fixed problem size, to the assumption of a fixed run time for each task. Indeed, researchers tend to adapt the size of the problem in order to solve it in a fixed amount of time, taking full advantage of the available resources. Hence, speedup is computed in terms of workload instead of execution time:

Equation 3.6: Workload-based speedup formula

$$S = \frac{W(N)}{W(1)}$$

Using the same approach as for Amdahl's law, the workload W is decomposed into parallelizable and non-parallelizable parts:

Equation 3.7: Parallelized workload

$$W(1) = W = (1 - \rho)W + \rho W$$

With parallelization by N processors, the parallelizable part can handle a workload of ρNW , for the same runtime. Hence the workload evolves linearly with parallelization to yield Gustafson's law formula:

Equation 3.8: Gustafson's law derivation

$$\begin{aligned} S &= \frac{W(N)}{W(1)} \\ \Rightarrow S &= \frac{(1 - \rho)W + \rho NW}{W} \\ \Rightarrow S &= (1 - \rho) + \rho N \end{aligned}$$

A comparison of Gustafson's law with Amdahl's law is proposed in Figure 3.3.

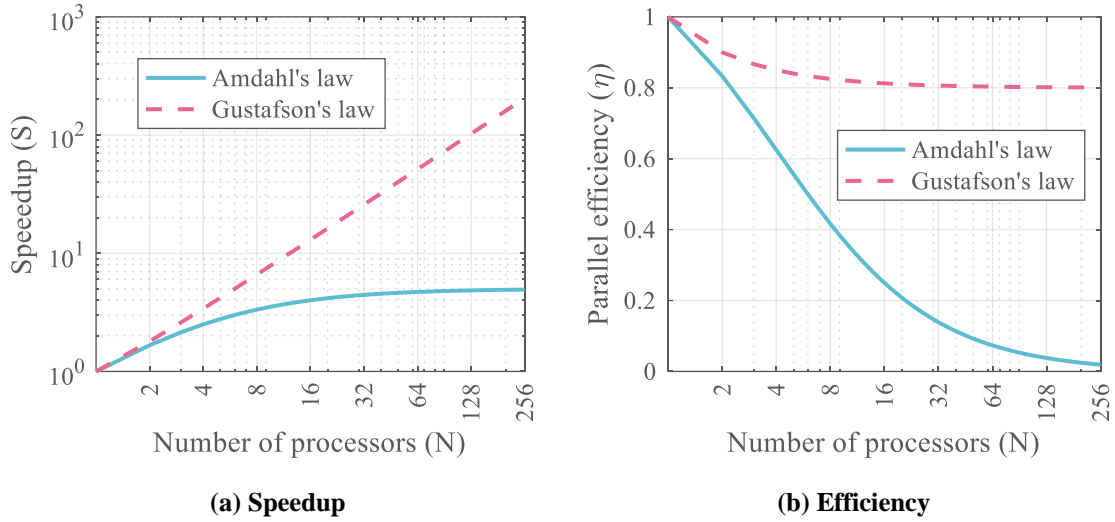


Figure 3.3: Gustafson's law compared to Amdahl's law

As expected, the speedup follows a linear trend with the number of processors while Amdahl's law reaches a limit. As for efficiency, Gustafson's law predicts that efficiency will be limited at 80%, corresponding to the proportion of parallelizable tasks. On the other hand, the efficiency given by Amdahl's law tends to zero.

When the serial process is divided into several heterogeneous parts, each with different capabilities, the previous quantities are adapted to account for this increased level of detail. Reasoning on execution time, each task i is represented with a proportion ρ_i of the execution time and is allocated a number of processors N_i from the parallelization process. A non-parallelizable part is then modeled with $N_i = 1$. The total execution time can now be decomposed as $T = \sum_{i=1}^N \rho_i T$ with $\sum_{i=1}^N \rho_i = 1$ and thanks to parallelization, each task takes $\frac{\rho_i T}{N_i}$ time. From these considerations, the speedup can then be derived:

Equation 3.9: Heterogeneous Amdahl's law speedup

$$S = \frac{T(1)}{T(N)} = \frac{T}{\sum_{i=1}^N \frac{\rho_i T}{N_i}} = \left(\sum_{i=1}^N \frac{\rho_i}{N_i} \right)^{-1}$$

A similar analysis leads to a reciprocal formula using Gustafson's law for massively parallelized processes:

Equation 3.10: Heterogeneous Gustafson's law speedup

$$S = \sum_{i=1}^N \rho_i N_i$$

A toy example can be studied to get a grasp of the behavior of speedup for heterogeneous systems. The following problem is considered:

A serial task is fragmented into four consecutive processes with execution time proportions being 7%, 23%, 12% and 58%. Assume that each process can be individually accelerated up to 20 times thanks to parallelization.

- 1. Use Amdahl's law to derive the speedup of the total system when each single process is being parallelized while the others are being kept serial.*
- 2. Assuming that the processes are respectively sped up 1, 5, 20 and 2 times, derive the speedup for the whole system.*

Solution: referring directly to Equation 3.9, the speedup of the system can be written as:

Equation 3.11: Solution of the parallelization toy example

$$S = \frac{1}{\frac{0.07}{N_1} + \frac{0.23}{N_2} + \frac{0.12}{N_3} + \frac{0.58}{N_4}}$$

Letting N_i changing from 1 up to 20 and letting the others constant at 1, the effect of the parallelization of process i can be isolated (Figure 3.4).

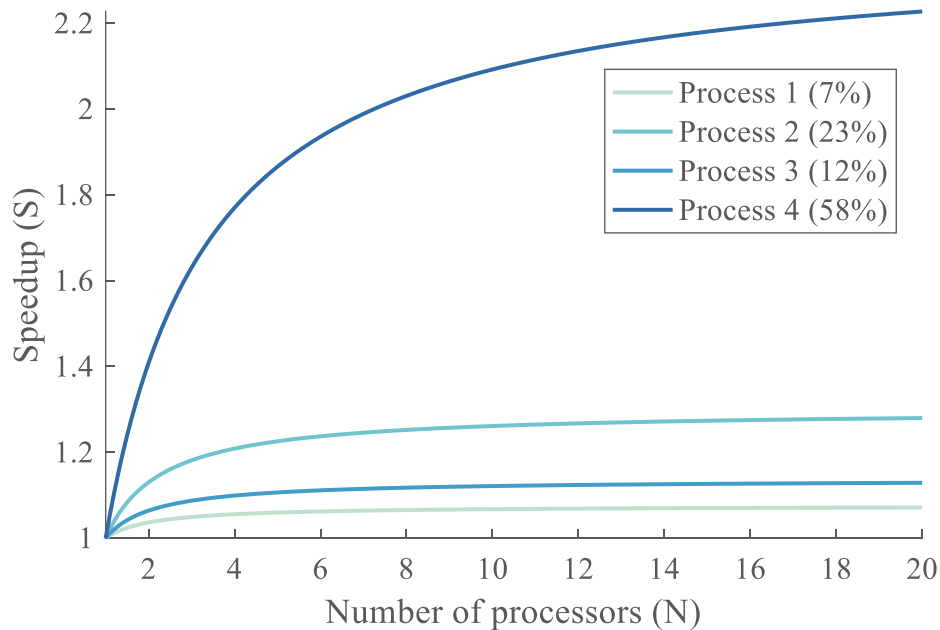


Figure 3.4: Heterogeneous parallelization

Due to its important weight in the execution time of the system, the fourth process seems to be dominant as its parallelization yields speedups which are well above the ones of the other processes. Using $N_1 = 1$, $N_2 = 5$, $N_3 = 20$, and $N_4 = 2$, the speedup of the

complete task is expected to be $S = 2.43$. This confirms the dominance of the fourth process as the overall speedup is very close to the individual speedup of process 4, despite the tremendous parallelization of processes 2 and 3. This type of insight is similar to conclusions drawn for systems in which overall performance is dictated by the least performant system (see introductory example of section 2.1).

While the previously described quantities are introduced for the field of parallel processing, similar concepts are applied to various fields including manpower management in software development: adding more persons to a late project ends up slowing it down instead of speeding it up [203]. Likewise, parallel slowdown can be extended to multi-robot systems: it seems right that for a very high number of agents, adding one entity will result in a similar, if not worse for some cases, performance for the swarm. This is in contrast with a very low number of agents, when adding two additional agents to a swarm of two entities may potentially occasion a tremendous improvement in performance. This notion tends to be quite the reverse of a commonplace intuition that adding more agents to a group of robots will necessarily result in a better performance.

Continuing on the introductory example, a parallel slowdown is experienced due to the small intermediate distances travelled by each agent (or more rigorously by agent N) between the sub-mapping missions. Indeed, each unit has to reach its initial mapping position and then return to it once the sub-mapping is done. A parallel can be drawn between these mission segments and the communication delays experienced in parallel computing: they represent the bottlenecks deteriorating the parallelization process. This

slowdown can be observed on the speedup and efficiency computed for the optimal design velocity of example 1 (Figure 3.5).

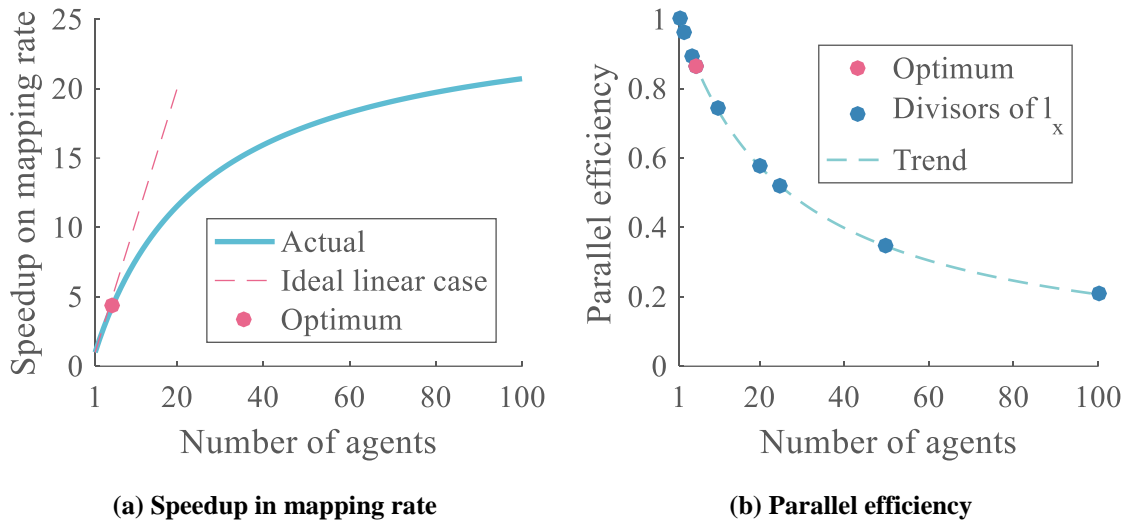


Figure 3.5: Limits of parallelization for the introductory example

The benefits of numerosity are less pronounced as the number of agents increases from one to a hundred. As expected, the speedup curve is below the linear ideal case: having twenty agents does not make the swarm map the area twenty times faster. This fact appears also on the parallel efficiency curve where the optimal solution in terms of cost lies at 86.61 % parallel efficiency. Note that for the problem setup presented earlier, the analytical solution makes sense only when the number of agents N divides the width l_x of the map (blue circles on Figure 3.5 (b)). If we assume that the even pattern is always used for any N , which translates into having some overlap between the allocated mapping areas, a general trend (dashed grey line) can be obtained.

Although the curves exhibit an almost asymptotic behavior, no clear limit in efficiency or in speedup is observed on Figure 3.5 as for the theoretical examples given on

Figure 3.2. Indeed, as opposed to the derivation of Amdahl's law, the proportion of the process which is parallelizable is not constant for this mapping example. The distance the furthest agent has to travel is directly linked to $\frac{l_x}{N}$ and thus depends on the number of agents. Furthermore, this simplistic example does not model the limits encountered in real-world applications for the parallelization of such a process: mapping time stochasticity, communication delays, or the traffic and clustering of agents due to saturation of the available space.

Another important observation is that the parallelism efficiency does not depend on microscopic variables for perfectly homogeneous swarm and missions. Indeed, if each agent is assigned a similar task, the time to complete the mission mainly depends on the annex tasks each one of them has to perform. Hence, parallel efficiency is a way of assessing the quality of the parallelization process quite independently of the performance of each agent of the swarm.

3.1.1.2 The limit of parallel effectiveness

While the preceding performance metrics quantify the variability of a given optimal swarm design, an additional measure is required to try to measure how close a design is to a possible limit in parallel performance. This absolute limit may correspond most of the time to an unconstrained optimum with respect to the other metrics and may possibly represent a physical barrier to further improvements.

Such a concept can be found in the field of controls and automation theory in which response times are defined with respect to margins [2]. For the study of first-order, linear,

and time-invariant systems, the time constant τ is used to characterize the response to a step input (Figure 3.6 (a)). This constant is rigorously defined from the differential equation satisfied by the system:

Equation 3.12: First order system equation

$$\frac{dV}{dt} + \frac{1}{\tau}V = f(t)$$

With V the response and f a forcing function on the system. For such a definition, it happens that after a time τ , the response to a step input has reached 63% of its final value and 95% after a time of around 5τ . This relates to exponential decay for which the half-life, time to reach 50% of the final value, is $\tau \ln(2)$. For second order systems, there is no similar analytical relation of such response times with the design parameters and such relations are documented in abacuses. However, the time for the response to reach and stay within the 5% band of its final value is defined as $tr_{5\%}$ (Figure 3.6 (b)). An alternative response time is the rise time t_r , time to first cross the final value (Figure 3.6 (b)).

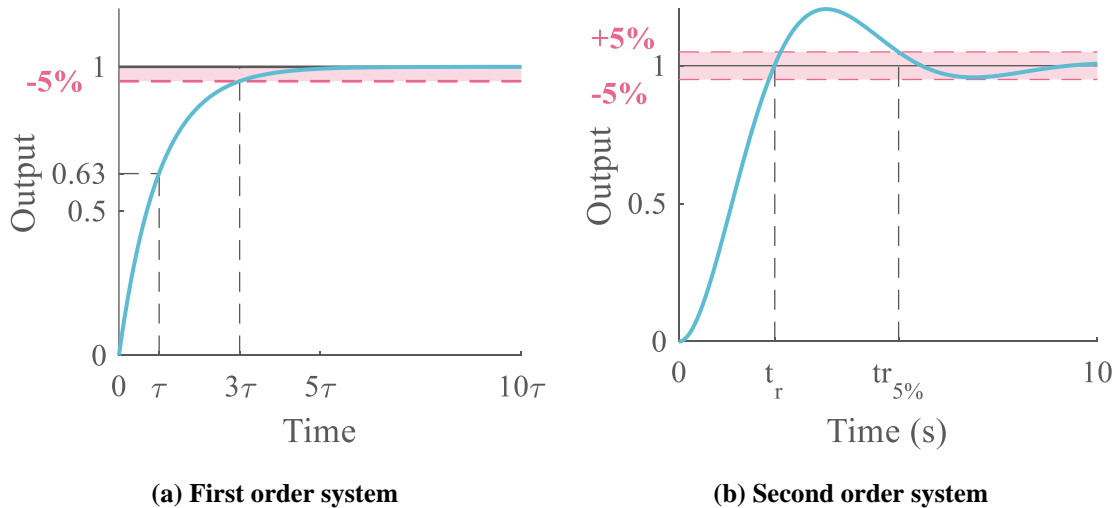


Figure 3.6: Controlled systems characteristic times

Henceforth inspired by the definition of time constants in the field of controls and automation, the concept of LOPE is introduced here. However, as opposed to stable automated systems used to define τ , $tr_{5\%}$, and t_r , there is no guaranty that the responses of swarm systems are stable with respect to the considered design variables (see Figure 3.5 (a)). Hence, incremental relative changes are used and the following quantities are introduced:

- $LOPE_{5\%}$: The value of the design variable (while all other variables are fixed) for which the incremental response falls and stays within 5%. While based on the insights of controls theory and the introductory example, this value of 5% is quite arbitrary and may depend on the design variable being considered.
- $LOPE_{\frac{2}{3}}$: The value of the design variable (while all other variables are fixed) for which the response is two thirds of its value at $LOPE_{5\%}$. This limit of parallel effectiveness is inspired by $\tau_{63\%}$ of first order systems to give an insight to the designer of when two thirds of the parallel effectiveness have been consumed in the design variable.
- $LOPE_{\frac{1}{2}}$: The value of the design variable (while all other variables are fixed) for which the response is half of its value at $LOPE_{5\%}$.

Although these quantities make more sense when computed with respect to the number of agents so as to remain directly linked with parallelization, they can be adjusted to fit a particular problem. They are better visualized as shown on Figure 3.7 and Figure 3.8 which are based on the introductory example.

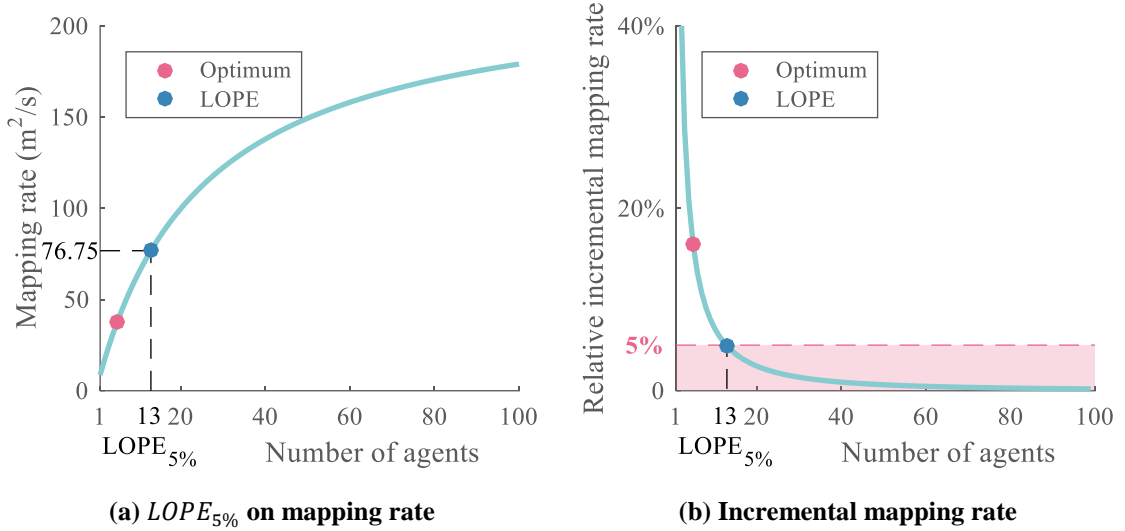


Figure 3.7: Visualization of LOPE for the introductory example

The $LOPE_{5\%}$ is 13 agents for a value of the mapping rate of $76.75 \text{ m}^2/\text{s}$ (Figure 3.7 (a)). It is the number of agents for which the incremental mapping rate falls and stays within 5% (Figure 3.7 (b)). The computation of the incremental mapping rate is explained on Figure 3.8 where the incremental mapping rate is of 5.47% at 12 but 4.92% for 13, the $LOPE_{5\%}$.

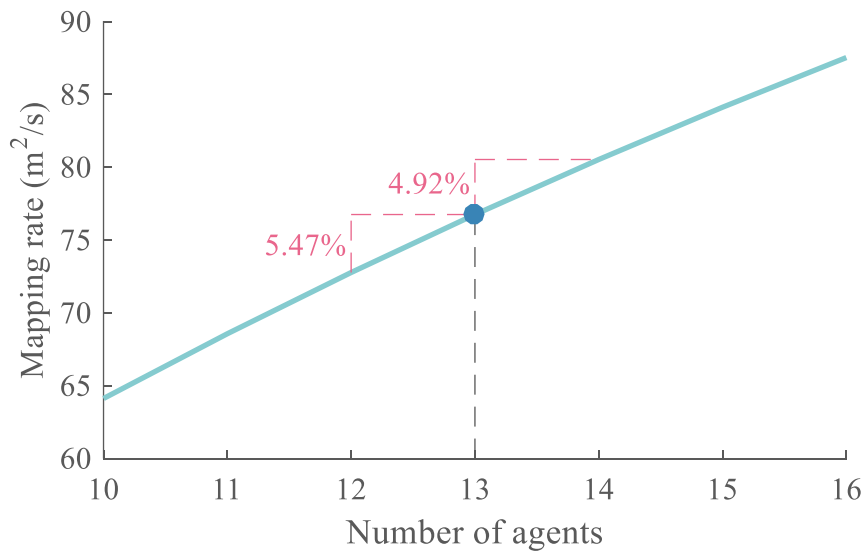


Figure 3.8: Detail of LOPE at 5% for the introductory example

All limits of parallel effectiveness for the introductory example are then displayed on Figure 3.9.

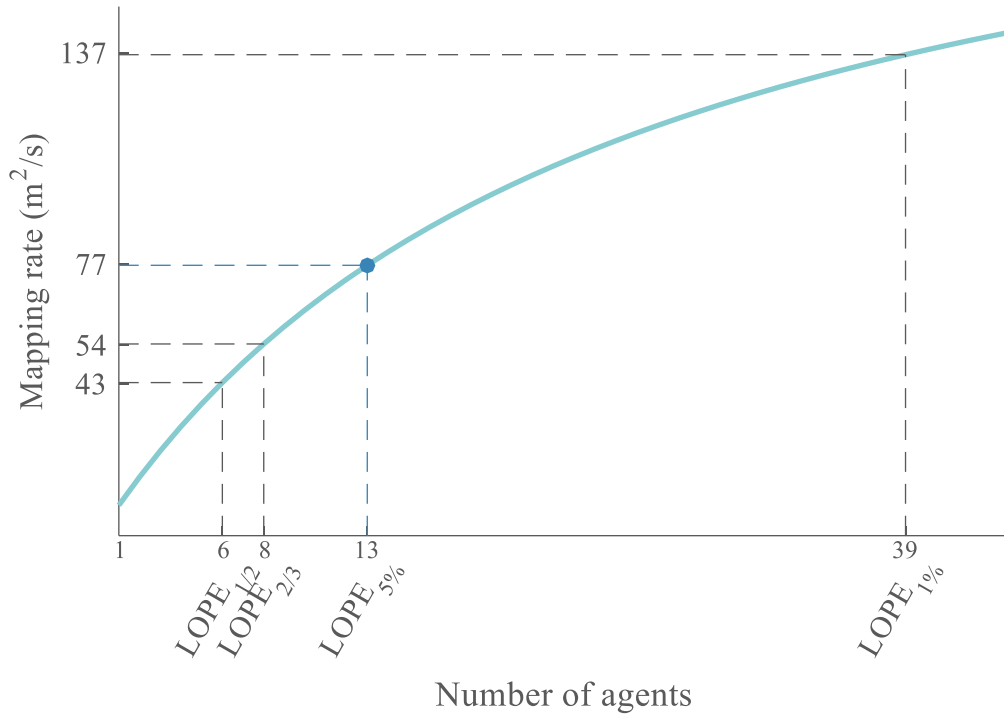


Figure 3.9: All LOPE quantities for the introductory example

The $LOPE_{\frac{1}{2}} = 6$ and $LOPE_{\frac{2}{3}} = 8$ tell the designer how far a design is from $LOPE_{5\%} = 13$. It can also be seen that $LOPE_{1\%}$ is far more demanding to attain in resources as its value sits at 39 agents in the group compared to 13 for $LOPE_{5\%}$. This quantity lets the designer know that after 39 agents in the system, the performance will not improve by more than 1% per additional agent.

3.1.2 Introduction of marginal quantities

Building up on the introductory example of the 2D mapping swarm, a sensitivity analysis can be carried out in the vicinity of the optimal design point. A sensitivity analysis helps in the design process in making sure that the obtained optimal design is somehow robust to slight variations in the design parameters. These latter may change due to evolving requirements or as a result of the manufacturing process for instance. Figure 3.10 presents such sensitivity plots around the optimal design point derived in the introductory example.

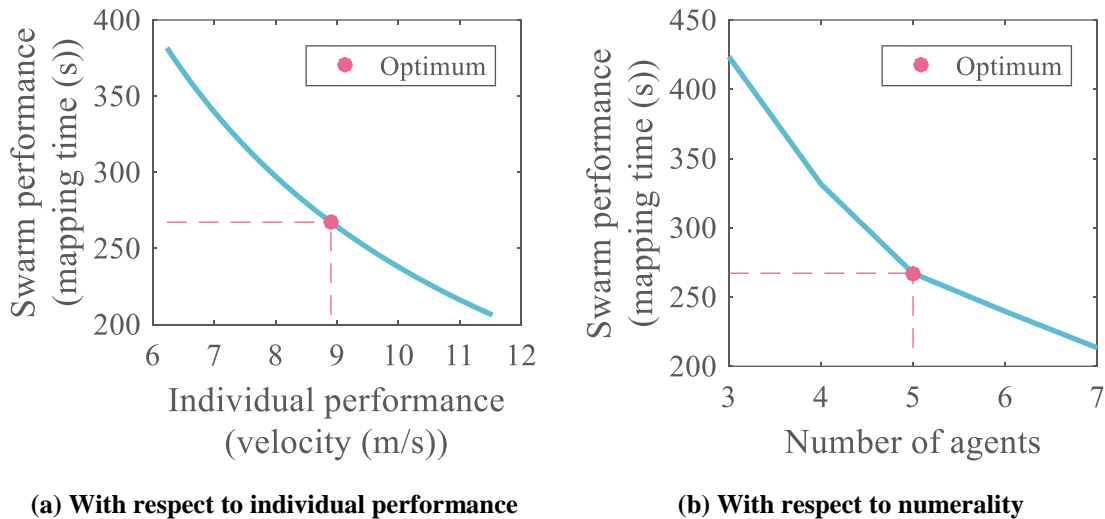


Figure 3.10: Sensitivity analysis around optimum design

The sensitivity plots confirm the general trend observed on the whole design space: an increase in the individual performance of the agents or in the number of agents results in an improved group performance. These prediction profilers enable the exploration of cross sections of the response across the different design factors. They represent the variation in the response when one variable is changed while the others are held constant

at the optimal values. The steepness of the curve hence enables to assess the importance of each factor.

In particular, it is important for the designer to try to quantify these variations around the optimum so as to measure the robustness of the design or predict which design variable will have the most impact on the response with the least cost variation. Such a variation can be quantified owing to the partial derivative of the response with respect to the considered design variable. Using the notations from the introductory example, the variation of the mapping time T with respect to the number of agents N in the swarm can be quantified around the optimum as $\left. \frac{\partial T(v, N)}{\partial N} \right|_{v^*, N^*}$: the partial derivative evaluated at the optimum design point (v^*, N^*) . The use of such quantities spreads to the field of optimization as a way to evaluate the response cost of violating the optimization constraints. They are also utilized in economics under terms such as marginal cost or shadow price. The first one refers to the cost incurred by producing one more unit of a good while the latter, although equal, is rather used to talk about the maximum price a decision-maker would be ready to pay to produce one more unit of good.

Based on this theory and the previous considerations, marginal quantities which are adapted to the design of swarming systems are proposed here as performance metrics. In order to quantify the commonplace statement ensuring that more agents result in enhanced performance of the swarm, these marginal measures are computed with respect to the number of agents in the group in the first place. Their formulation is then extended to other design variables, microscopic or macroscopic. The following quantities are thus introduced

and described hereafter: Marginal Group Performance (MGP), Marginal Group Cost (MGC) and Marginal Group Efficiency (MGE).

Marginal Group Performance: principal performance metric introduced here, the MGP originally aims at putting numbers on the ordinary thought that by increasing the number of agents in the swarm, the performance will improve. Fundamentally, the MGP represents the variation in group performance if the number of agents is increased by one. By extension, similar quantities can be derived with respect to other design variables. The MGP would then represent the deviation of group performance when a unit change occurs on design variable χ :

Equation 3.13: Marginal Group Performance

$$MGP_{,\chi} = \frac{\partial P}{\partial \chi}$$

Going back to the initial motivation for the introduction of the MGP, the effect of numerality on the performance of the optimal design is quantified by:

Equation 3.14: Numerality Marginal Group Performance

$$MGP_{,N} = \frac{\Delta P}{\Delta N}$$

Marginal Group Cost: While the MGP focuses on variations in the performance of the system by unit changes in the design variables, the MGC quantifies at what cost such

changes are made. Hence, the MGC is directly complementary to the MGP to the eyes of the designer. In the same way that the MGP is defined, the MGC can be written as:

Equation 3.15: Marginal Group Cost

$$MGC_{,\chi} = \frac{\partial C}{\partial \chi}$$

Marginal Group Efficiency: finally, in a similar fashion to which the MGP is originally conceived, the concept is to capture the variations in parallelism efficiency resulting from a change in the number of agents in the swarm. This idea mainly stems from the fact that parallelism efficiency of a system is often most affected by the number of agents composing it. However, this definition can be extended to design variables other than numerality:

Equation 3.16: Marginal Group Efficiency

$$MGE_{,\chi} = \frac{\partial \eta}{\partial \chi}$$

As it was previously mentioned, the parallelism efficiency does not depend on microscopic variables for specific homogeneous swarms and missions. Hence, the marginal group efficiency will likely be zero when computed with respect to such variables.

These performance metrics are evaluated around design points to be studied. Being derivatives of the system responses, they provide the designer with clear insight on how the system performance is changing around the chosen optimum design, and more generally in the design space. Going back to the first introductory example, the marginal quantities around the optimal point are presented in Table 3.1 with performance P_1 being the mapping time and performance P_2 being the mapping rate.

Table 3.1: Marginal quantities for the introductory example

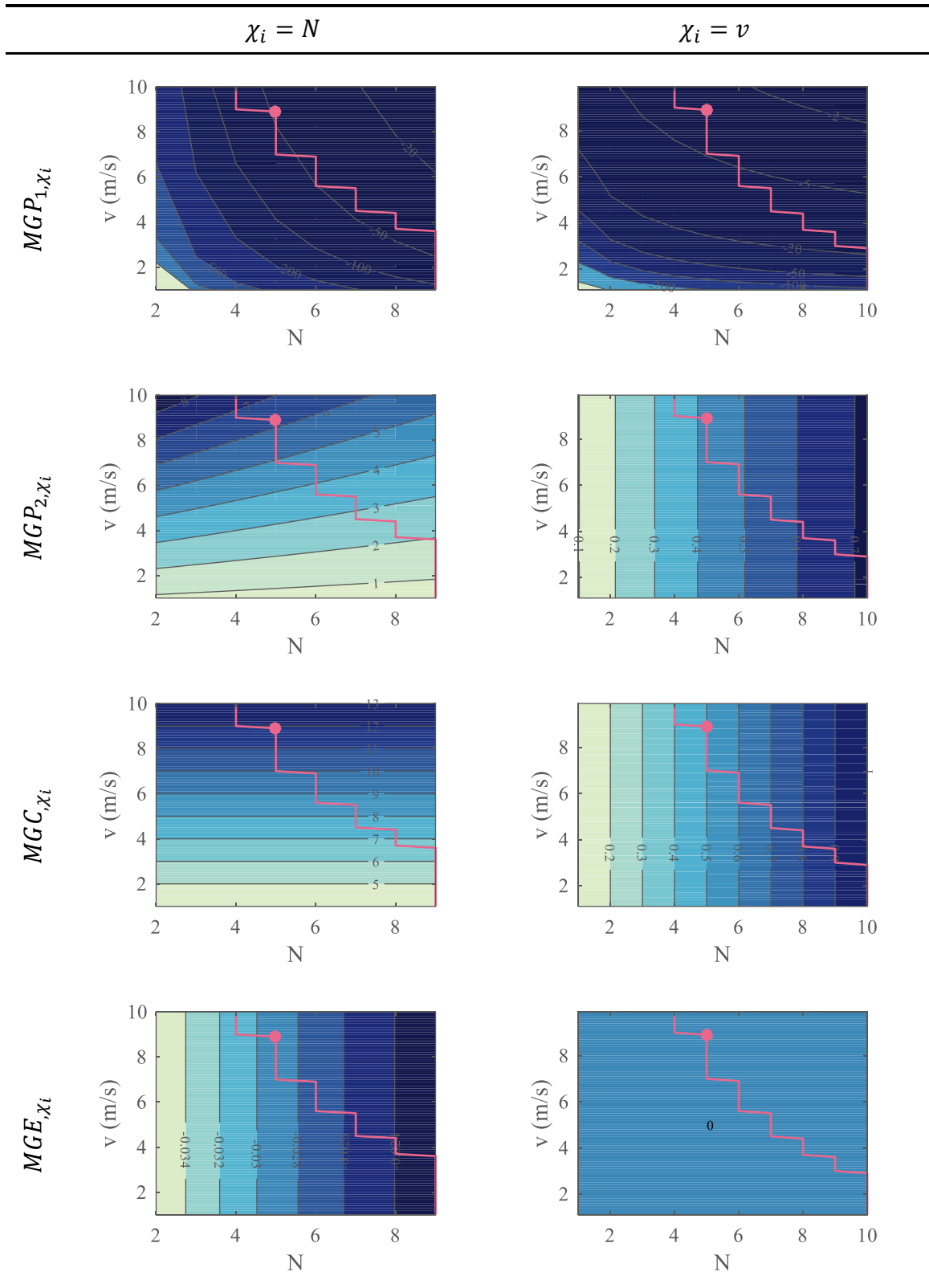
Performance Metrics	Design variables	
	Macroscopic $\chi_i = N$	Microscopic $\chi_i = v$
MGP_{1,χ_i}	-45.97 s	-3.00 s/(m/s)
MGP_{2,χ_i}	5.79 m ² /s	0.42 (m ² /s)/(m/s)
$MGC_{,\chi_i}$	11.90	0.50 (m/s) ⁻¹
$MGE_{,\chi_i}$	-3%	0 (m/s) ⁻¹

By displaying the design variables column-wise, the designer can quickly identify which variable is having the most impact on any response. Considering the introductory example, adding one agent to the mapping swarm would decrease the mission time by 45.97 s, however it would also increase the cost by 11.90 units and deteriorate the parallelism efficiency by 3%. On the other hand, it can be seen that the individual performance of the agents as a lesser effect on the mapping time but also for a smaller cost impact. Indeed, increasing the velocity of the agents by 1 m/s results in a mapping time decrease by 3 s and a cost increased by 0.5 only. Note that as explained earlier on and as

expected for this particular mission, $MGE_{,v} = 0 (m/s)^{-1}$: the individual performance of the agents does not have an impact on the parallel efficiency.

The marginal quantities can also be plotted over the design space to understand how the derivatives of the performance metrics vary with the design variables. Table 3.2 exhibits these variations with the rows representing the marginal quantities while each column denotes the design variable considered for the computation of a marginal quantity. As for the previous contour plots of the introductory example, the red line represents the cost constraint and the red star is the previously derived optimal design point. These graphs should be understood in such a way that, by fixing a velocity v and reading different values of N on a horizontal line, the designer is contemplating the following question: “with swarm agents having this velocity v , what increments in my performance metrics are possible by adding one more agent?”. By looking up these marginal values for different values of N , the designer gains insight to figure out at what size of the swarm it is not beneficial anymore to add more agents. Same goes for the second column and adding velocity to the individual agents.

Table 3.2: Marginal quantities over complete design space



Firstly, the marginal group performance deteriorates with both N and v , confirming the limits of parallelization. In fact, with more and more agents in the swarm, the benefits of adding one agent to the swarm ($MGP_{1,N}$) or 1 m/s to the velocity of each agent ($MGP_{1,v}$) are not as flagrant as for a swarm with fewer individuals. This is deduced by looking at the horizontal axes of the MGP contour plots. The same conclusion is drawn with the vertical axes: for swarms with very performant individuals, the benefits of adding one agent to the swarm or 1 m/s to the velocity of each agent are lesser compared to the potential benefits obtained on a swarm with slow agents. The contour plots of $MGP_{2,N}$ and $MGP_{2,v}$ show the same trend on the mapping rate. $MGE_{,N}$ provides a good insight on the variations of incremental parallelism efficiency: with a swarm of 5 agents, adding an agent deteriorates the parallelism efficiency by 2.9% while this latter is affected by only 1% in a swarm of 25 agents.

A second observation is that due to the linearity of the swarm performance metrics, some of the marginal quantities do not depend on certain design variables. Indeed, noting that from Equation 2.7 $T(v, N) \propto \frac{1}{v}$, then $R(v, N) \propto v$ and consequently $MGP_{2,v} = \frac{\partial R}{\partial v} \propto 1$ is independent of variable v . Using a similar approach, it can be verified that $MGC_{,v}$ and $MGE_{,N}$ do not depend on v , and $MGC_{,N}$ does not depend on N : observations in correlation with what is observed in Table 3.2.

Finally, for all contour plots, the amplitude is a lot less for the marginal quantities computed with respect to v . Hence, between the two design variables N and v , it seems that for this case the number of agents has much more impact on the metrics than the velocity of the agents. Moreover, $MGE_{,v} = 0$ over the whole design space as expected since

in this particular mission of a homogeneous swarm, the parallel efficiency does not depend on the individual capabilities of the agents.

This type of graph clearly shows the limits of parallelization for the introductory example and motivates the designer to know up to what point the parallelization should be preferred over individual performance improvements, and vice-versa. While parallelization efficiency and marginal quantities have been established and quantified in the preceding sections, no margin measure was defined to systematically obtain an optimum configuration. The optimal solution for the introductory example was based on the maximization of the performance with respect to a constraint of cost but one could wonder if the optimization could be carried out in terms of parallelization efficiency for instance. The next section establishes possible metrics towards a benchmark evaluation methodology for multi-robot systems.

3.1.3 Benchmarking

The lack of an established design framework for multi-robotics stems to a great extent from the lack of a reference benchmarking mission. Such a standard would have to cover a vast spectrum of possible applications in multi-robotics: collective exploration, coordinated motion, collective transport, self-assembly, chain formation, consensus achievement, and many more [95]. A fundamental error resides in the fact that a universal benchmarking mission or design framework, tailored to various types of applications, is an evident oxymoron. A customizable universal design framework can however be conceived.

Using the generic top-down design decision support process presented on Figure 3.1, the generated alternatives have to be evaluated for further comparison. For the framework to be customizable, it needs to be applicable to very different mission types without privileging the canonical mission used in this work. Table 3.3 presents several types of missions typically utilized with robotic swarms and the pertinent metrics used to evaluate these missions.

Table 3.3: A disparity of metrics

Mission	Metrics
Search and Rescue [117]	Victim tracking effectiveness Time to complete objective Maintenance of communication links Energy consumption
Oil Spills Detection [117]	Percentage of polluting vessels found Average time to identify a polluting vessel Communication links maintenance
Mapping	Time to complete the map Quality of the final map Number of maps reconnection issues Cost of the swarm system Distance traveled by robots [204] Total explored area Loop closure capabilities

As observed in Table 3.3, there is no normalized metric enabling the evaluation of the different types of mission which could be used in the proposed methodology. Some missions do not even have the same number of metrics to characterize them. In order to be able to use the methodology for various swarm missions, several utility functions are proposed which aggregate the multiple performance attributes of the system depending on

their pertaining to a given category: execution, quality, and cost. This division is based on widespread paradoxes observed in many design disciplines and in nature: “quantity versus quality”, “fast versus strong”, “rapidity versus stability versus precision”, and “performance versus cost” tradeoffs. By this phrasing, it is meant that in many occurrences these objectives tend to be in conflict and are hence considered separately. For instance, in the response of automated systems, rapidity is often achieved at the cost of stability or precision of the response [2]. The overall evaluation criteria are defined as the following.

Execution index: indicative of the pure performance of the system, this index regroups all mission metrics relating to the “quantity” achieved during the mission of the system. For example, for the introductory mapping example the execution index would be a combination of the mapping time and the mapped area (Table 3.4).

Equation 3.17: Execution index

$$\varepsilon = \sum_{M_i \in \left\{ \begin{array}{l} \text{measures} \\ \text{of execution} \end{array} \right\}} \alpha_i M_i$$

Completion or quality index: sometimes ambiguously linked to the execution index, the completion index represents the “quality” (as opposed to quantity) achieved during the mission of the system. It regroups all metrics assessing the degree of completion of the mission. Reminiscing the introductory example, a completion index for a mapping mission may be a combination of the distance errors on the map landmarks, the density of the final map, and the number of sub-maps reconnection errors (Table 3.4).

Equation 3.18: Completion index

$$Q = \sum_{M_i \in \{\substack{\text{measures} \\ \text{of quality}}\}} \alpha_i M_i$$

Cost index: complementary to the two previous performance metrics, the cost index regroups all metrics linked to the cost of the system, be it in money, energy, time, or raw materials. Continuing on the canonical example, the cost function can simply be the cost of the swarm (Table 3.4).

Equation 3.19: Cost index

$$C = \sum_{M_i \in \{\substack{\text{measures} \\ \text{of cost}}\}} \alpha_i M_i$$

Such aggregate functions are key enablers for the modularity of the proposed design methodology as they can potentially accommodate any swarming mission type. Note that it is possible to use the OEC approach described in [141] in order to quantify the performance with respect to established requirements or baseline values. Put into perspective, these metrics enable the assessment of the absolute performance of a system, how well this performance was achieved, and for which cost. Note that these metrics are not always completely uncorrelated and it is the choice of the designer to segregate the metrics according to design preferences. The versatility of this approach is demonstrated in Table 3.4 where the metrics used for each index are separated. The α_i coefficients are left to be defined.

Table 3.4: A set of unified metrics

	Execution Index	Completion Index	Cost Index
Chain formation	<ul style="list-style-type: none"> • Time to complete chain 	<ul style="list-style-type: none"> • Proportion of missed links 	<ul style="list-style-type: none"> • Cost of swarm • Energy consumption • Number of unused agents
Mapping	<ul style="list-style-type: none"> • Time to complete map 	<ul style="list-style-type: none"> • Density of map features • Percentage of reconnection errors 	<ul style="list-style-type: none"> • Cost of the swarm system
Search & Rescue	<ul style="list-style-type: none"> • Percentage of victims saved • Time to complete objective 	<ul style="list-style-type: none"> • Victim tracking effectiveness • Maintenance of communication links 	<ul style="list-style-type: none"> • Energy consumption

After the different metrics from Table 3.3 are combined into the aggregate functions, the diverse missions can now be evaluated in the design methodology with the same measures, hence facilitating the use of the framework for different missions.

This approach, also known as a priori multi-objective optimization assumes that the designer is able to rationally decide on the value of the weighting coefficients for each original metric. Aggregate functions also have the advantage of facilitating the optimization process with the use of single objective optimization methods compared to a posteriori techniques. This feature is essential since, due to the extremely high number of alternatives to evaluate, rapidity of the code is essential. By reducing the dimensionality of the problem, these indices also simplify the decision-making process which can get quite delicate with multiple objectives. These characteristics further motivate the choice of a priori multi-objective optimization as established in section 2.3.2.1.1.2.

In this section, performance metrics pertinent with the problematics at stake have been introduced. Notions of parallelism efficiency are first presented to quantify the effects of parallelization on the system. In addition, by analogy to controls theory, a few metrics are formed to provide the designer with an absolute measure of when the effects of parallelization are starting to vanish in the performance of the system. A second set of metrics introduced as marginal quantities offer a systematic approach to quantify the sensibility around a given design point in terms of performance, parallelization, and cost. Finally, to promote modularity and reusability of the proposed method with various swarm mission types, aggregate functions are proposed. These metrics are created based on three complementary aspects of the mission of a swarm: its quantity, its quality, and its cost. The conjunction of these performance metrics provides the means to compare different swarm alternatives and take decisions.

3.2 Design space definition

The definition of the design space consists in establishing the boundary of the set of feasible alternatives which are to be compared and optimized.

3.2.1 The design variables

The choice of design variables is a key step in the methodology as their number and their range greatly affect the size of the design space which is already amplified due to its multi-level and multi-architecture nature. For multi-robotics, design variables intervene in microscopic and in macroscopic levels, hence both need to be detailed in the present subsection.

3.2.1.1 Microscopic level: the agents

Given that the study of heterogeneous swarms must be incorporated in this work, agents must have the opportunity to be quite different in architectures and capabilities. To differentiate the different architectures to be considered, a first taxonomy of UAVs inspired by [205] is proposed in Figure 3.11. It first segregates Heavier-Than-Air (HTA) aircraft and Lighter-Than-Air aircraft (LTA). Note that unpowered aircraft such as gliders, sailplanes, balloons, and aerostats, are not considered as their performance on the benchmarking mission would prove dependent on variables such as wind or thermals which require a detail of modeling beyond the scope of this research. Such unconsidered architectures are greyed out in Figure 3.11. Moreover, the microlight category is not included since UAVs, already light by nature, do not have such a category.

Heterogeneity in architectures is attained if there are at least two different architectures to be considered in the design space. However to ensure generality, at least four architectures will be considered in this research: fixed-wing, helicopter (and more particularly multicopter configurations), dirigibles, and ornithopters. The considered architectures are detailed in hereafter with their definitions [206] and their design variables.

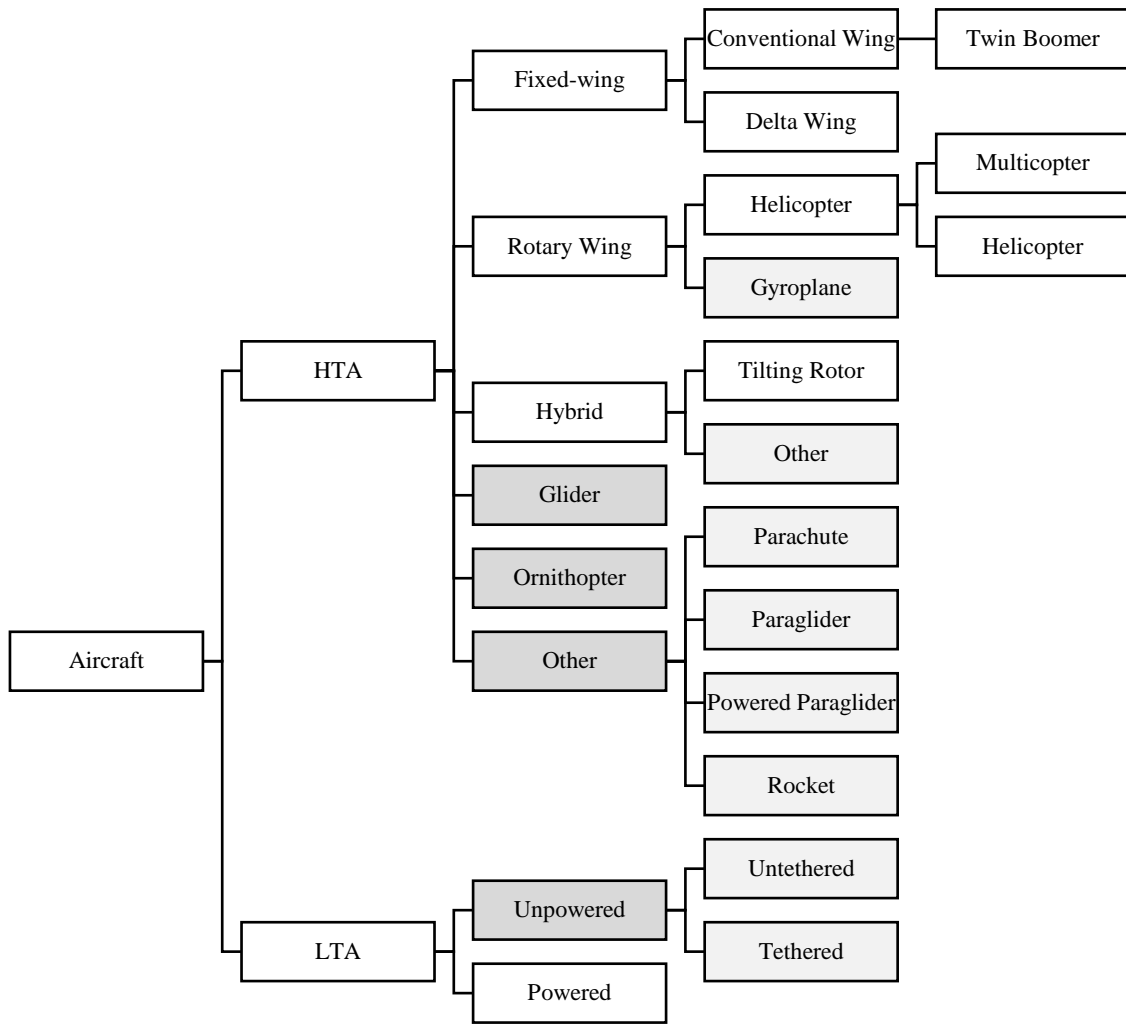


Figure 3.11: A taxonomy of existing UAVs

Fixed-wing: a fixed-wing airplane is a heavier-than-air aircraft with wings which remained in a fixed position under given conditions of flight. This category may include variable geometry aircraft [206]. Possible design variables include: wing type, wing span, wing aspect ratio, fuselage length, propellers diameter, type of energy storage system (battery or fuel), and embarked sensors and electronics.

Helicopter: A helicopter is a heavier-than-air aircraft supported in flight chiefly by the reactions of the air on one or more power driven rotors on substantially vertical axes [206]. Design variables which can be considered for helicopters include propeller diameter and type, length of the arms, energy storage system, as well as embarked sensors and electronics.

Dirigible: a dirigible is a power-driven lighter-than-air aircraft [206]. Possible design variables to be included for dirigibles include engine type, propeller type and diameter, and embarked sensors and electronics.

Ornithopter: an aircraft which flies by flapping its wings. Possible design variables include wing area, type of stabilizers, battery capacity, and embarked electronics.

3.2.1.2 Macroscopic level: the swarm

The group of robots is mainly represented with three pieces of information: the architectures involved, the number of agents for each architecture, and the type of control architecture used. Note that as a first approach, it can be assumed that it is not possible to combine the same architecture under different configurations in the same swarm. That is, each architecture is represented by only one configuration in the swarm. For instance, if the swarm contains quadrotors, then all the quadrotors in the group have the same characteristics. A notional summary of the design variables at the macroscopic level is proposed in the morphological matrix Table 3.5. The selected options are in green and incompatible options with the selected choice are highlighted in red.

Table 3.5: Notional morphological matrix at the macroscopic level

Features	Option 1	Option 2	Option 3	Option 4
Architecture	Quadrotor	Twin-boomer	Dirigible	Ornithopter
Architecture numerality	6	2		
Control architecture category	Centralized	Decentralized		
Control scheme	Leader/Follower	Hierarchical	Consensus	Distributed

The available options for each of the macroscopic design variables are detailed in the modeling section 3.3.

3.2.2 Alternatives generation

The generation of alternatives is to be conducted with respect to the design-space exploration methodology described in the second chapter and a quick overview of this process is proposed here as a reminder (Figure 3.12). A first step consists in the reduction of the dimensionality of the design space by applying the morphological reduction proposed by [173] (see page 142). Then, the whole morphological analysis is represented with the help of a tree, enabling to dynamically allocate alternatives depending on the choices made at the macroscopic level. The architecture level is an intermediate level containing the abstract morphological matrices of the considered architectures.

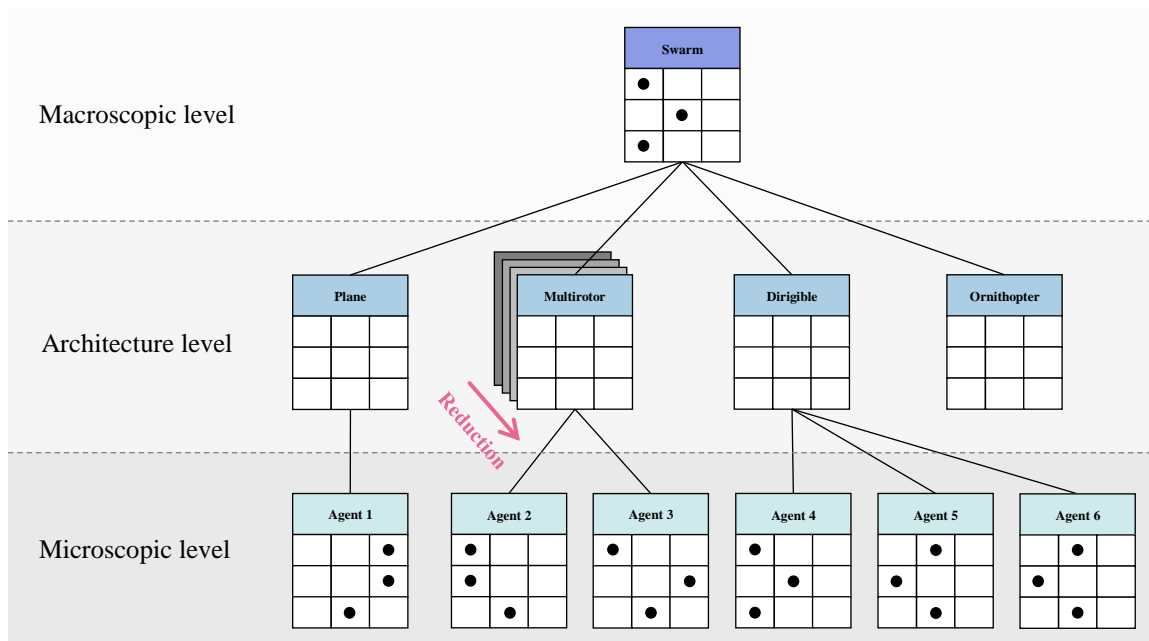


Figure 3.12: Review of hypothesis 3.1

3.3 Alternatives evaluation

Once an architecture to be studied has been given by the design space exploration process, its performance needs to be measured in terms of the established metrics with respect to the canonical mission. Based on the physics-based simulation approach motivated by the second chapter, the whole mission has to be carried out by the agents. In order for the performance evaluation environment to properly fit the complete methodology, important features to consider are the following:

- **Automation & Integration:** due to the very high number of cases to be run, the environment must be able to take as input designs which are automatically generated by the design space exploration method. Performance results must also be generated automatically.

- **Conceptual design level:** owing to the concentration of this work on conceptual design phases, a corresponding level of modeling should be used. In particular, advanced and detailed information about the agents of the swarm are not available at this design stage. For example, Computational Flow Dynamics (CFD) techniques requiring exhaustive shapes information are typically not utilized for this step. This would also entail a higher number of design variables and longer execution times.
- **Cost modeling:** in order to be able to place conflicting constraints on the system such as the prominent performance versus cost tradeoff presented in the introductory example, a minimum level of cost modeling should be implemented.
- **Physics-based:** due to the lack of historical data available in the design of swarming systems, a physics-based approach should be preferred. Moreover, a direct link between the configuration, especially geometry, of an agent and its performance must be established.
- **Rapidity:** owing to the very high number of cases to evaluate in the design space exploration step, a simulation environment with very small runtime shall be preferred.

As the research challenges underline (Figure 1.37 page 70), even a canonical problem with a limited number of design variables quickly generates a colossal design space suitable for this current work. Moreover, a restrained level of simulation detail would prove fast enough and emergent behaviors can be observed without any further complexity. Therefore, this section describes a proposed example of implementation for models which have to be established for the agents and for the swarm as well.

3.3.1 Microscopic level: the agents

The modeling of the different type of agents, and especially the required level of detail, is based on a top-down decomposition into subsystems (Figure 3.13).

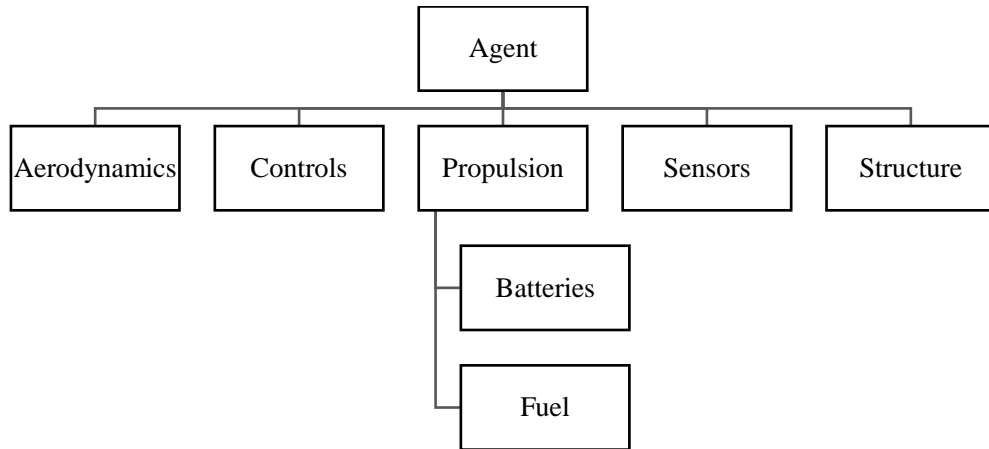


Figure 3.13: Agent modeling breakdown

The models to be implemented have to be moderately detailed and fast to evaluate in order to fit in the mesoscopic representation of the whole system. Details of modeling for a quadcopter are given in [207], and for a fixed-wing UAV in [208]. A conceptual design process for UAVs such as the one proposed in [209] is used: based on the physical breakdown of the agents, the different models are detailed here below.

- **Aerodynamics:** this part considers simplistic models of lift and drag and no high-speed aerodynamics effects are to be included for reasons detailed here below. The aerodynamics coefficients are obtained from current concepts in the literature or by extended vortex-lattice models such as AVL as recommended by [210].
- **Cost:** cost information for off-the-shelf components are included for the model.

- **Controls:** simple Proportional–Integral–Derivative (PID) controllers are to be implemented as they are widely utilized and understood. They facilitate the implementation as well as the tuning of the gains for each architecture.
- **Propulsion:** the propulsion systems are composed of motors and propellers for most of the architectures. For the modeling part, an extensive database of compatible motors and propellers is used with performance data.
 - **Batteries:** generic charge and discharge models are used.
 - **Fuel:** fuel consumption data is utilized for the rate of change of fuel mass.
- **Sensors:** sensors are precisely modeled in their behavior since they most probably are the main drivers of the performance of the agents.
 - **Lidar:** the lidar model uses a ray casting algorithm that returns the distance of the scanned environment points. The design variables are the resolution, the range, and the scan rate [211].
 - **Camera:** basic ray-casting is used to obtain the projected colors for every pixel of the image but no distance information is provided. Main design variables are the resolution, the framerate, and the color type (RGB or grayscale).
 - **RGBD-camera:** mix between the lidar and the camera, ray-casting is used in the same fashion as for the image returning both the color and the distance of the scanned environment point.

Note that measurement noise may be added for each of the sensors if required.

- **Structures:** structural effects are neglected in this work given the focus on micro-sized aerial vehicles where structures are most of the time sturdy enough for the

dynamics and the weights of the subsystems. Moreover, aeroelastic effects are neglected.

- **Weight estimation:** the computation of the weight of an agent is based on the off-the-shelf components database information as well as material properties and empiric relations for geometric features.

Finally, the dynamics of the agents are analyzed through a 6DOF model based on a flat Earth assumption. Indeed, the size of the mission area is assumed to be moderate enough so that non-inertial frame effects can be neglected (a couple hundreds of kilometers wide at maximum). A North-East-Down (NED) frame is used for reference.

For the given canonical mission, a good estimate of the key variables is obtained by considering possible elements that would limit the mapping performance. Given that the performance metrics on this mapping are mostly the time to map and the map quality, it is probable that significant drivers of the performance are going to depend on the speed at which the robot can travel, the update rate of the sensors, as well as their resolution.

It is expected that for agents of a moderate size (i.e. from nanodrones to micro-drones), the sensors are going to be one of the most limiting factors in the performance of the system. For instance, a quadrirotor with a span of fifty centimeters is easily able to reach high speeds above fifty kilometers an hour. However, due to its limited capacity to embark computing power and high quality sensors, its mapping rate might be relatively low and might constrain it to fly at low speeds. This justifies the fact that there is no need for a detailed modeling of high speed aerodynamics effects. Moreover, sensors often

represent quite an amount of weight and also imply an important current consumption, incurring the need of a bigger battery. Consequently, sensors are most probably the main drivers of the performance of an agent.

3.3.2 Macroscopic level: the swarm

Swarm modeling is a key element for this work due to its link with emergence of behavior and has to be carefully integrated into the mesoscopic model. This latter is based on a rather detailed modeling of the agents but a simple modeling of the rules of the swarm.

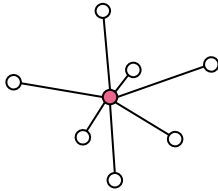
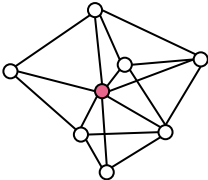
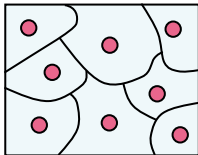
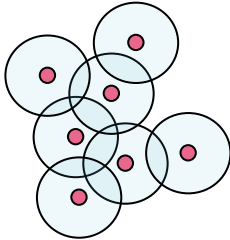
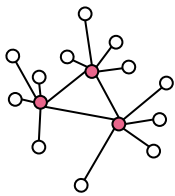
One of the main design variables for the macroscopic level is the type of control architecture. A handful of schemes has been developed and a comparison is proposed in Table 3.6. Such control schemes are first divided between centralized and decentralized methods. In the first category, a central unit which may be a ground station or a swarm agent itself, is in charge of the cohesion and the decision within the swarm. This access to global information leads to intrinsic coordination of the group and optimal results [212].

In the latter category, no such central leader exists and the swarm control is achieved by the artificial intelligence of each individual agent. In the consensus technique, each agent takes into account the information communicated by the others in a voting system. If the voting score exceeds a given threshold function, an action has to be taken by the agent. The partitioned method is particularly adapted for mapping purposes or other area-based missions since the space is separated in different areas. Each agent is then assigned a particular space and the communication between the agents insures that they remain in their area. In the distributed scheme, no communication is necessary since every agent is only capable of detecting the other agents and acts based on their behavior.

Finally, there exist a hybrid category representing a mix between centralized and decentralized approaches so that the control does not depend on one unique leader but on several sub-leaders scattered within the group.

The decentralized control schemes divide the complex behavior and decision tasks into individual parallel processes performed on each robot, this generally guarantees better flexibility and scalability of the swarm. However, while they lower the use of bandwidth by the system, they tend to be much more complicated to implement than a centralized architecture. Moreover, they are also inclined to more instability and are less predictable [115], making it difficult to obtain a coherent global behavior [212]. In addition, decentralized techniques imply that a mutual detection algorithm be implemented between individual agents. Finally, decentralized architectures present risks of losing units when they go out of the communication range of all other neighbors. In terms of implementation, the leader/follower, the hierarchical, and the partitioned approaches are the simplest and could fit the assumptions of the mesoscopic scale. In sum, the centralized control architecture seems to be the most widely used, the simplest to implement and corresponds to the mesoscopic description. In terms of implementation, the control architecture can follow guidelines from [213].

Table 3.6: Swarm control architectures

Architecture	Representation	Advantages	Limitations
Centralized			
Leader Follower		<ul style="list-style-type: none"> • Reliable and predictable • Quick swarm response possible 	<ul style="list-style-type: none"> • Vulnerable to loss of leader • Communication required between leader and every agent • Leader requires high bandwidth capability
Decentralized			
Consensus		<ul style="list-style-type: none"> • Lower bandwidth requirements • Robust decision-making • No critical nodes 	<ul style="list-style-type: none"> • Communication required between all agents • Not as predictable • Requires voting • Slower overall
Partitioned			
Distributed		<ul style="list-style-type: none"> • No communication required • Absolute position not required • No critical nodes 	<ul style="list-style-type: none"> • Difficult to reliably and quickly control • Requires all agents to be detected as near neighbors
Hybrid			
Hierarchical		<ul style="list-style-type: none"> • Fairly reliable and predictable • Loss of one leader is not catastrophic • Quick partial swarm response possible 	<ul style="list-style-type: none"> • Leader-follower communication required • Coordination required between sub-groups

An additional set of assumptions is hence established here below:

- Perfect communication between the agents, equivalent to an infinite communication range.
- No collisions are modeled. However, saturation of the mission space may happen if too many agents are deployed. This is represented by the area occupied by the agents when they fly at the same altitude.
- Limits of numerality are also represented by the computation and network bandwidth limits of the centralized entity.

This set of assumptions makes the swarm performance on a given mission deterministic. This liberates from the use of replications for each mission execution, consequently accelerating the model evaluation: a key requirement for mesoscopic representations. Moreover, this has the advantage of speeding up the design space exploration. For the same reasons, uncertainty beyond the marginal performance quantities is not considered in the scope of this work.

3.3.3 Agent-based simulation

An agent-based model is a computational model that enables the simulation of autonomous agents and their interactions, resulting in modifications of a system and its environment. Such models are particularly adapted for complex systems with nonlinear dynamics, heterogeneity, and emergence. As a matter of fact, “agent-based modeling postulates that the global behavior of a complex system derives from the lower level interactions of its constituent elements” [160]. This category of models hence seems

perfectly adequate for the scope of this research. The main modules of an agent-based model are the environment, the agents, and their interactions (see Figure 3.14).

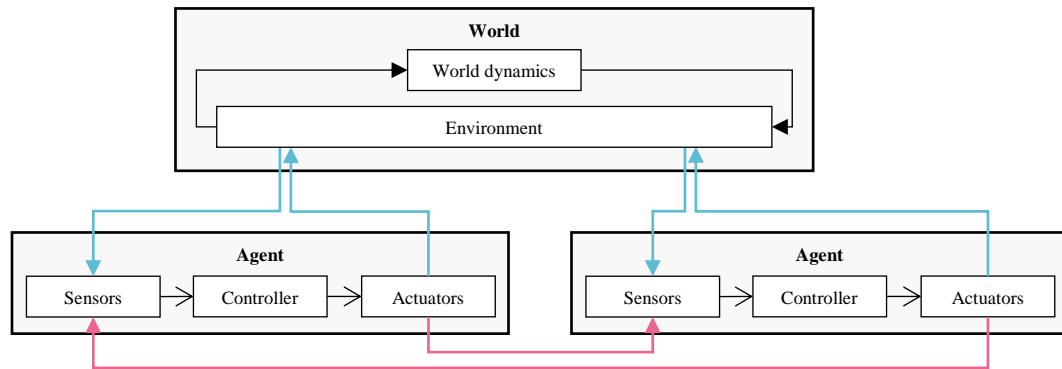


Figure 3.14: Proposed agent-based model architecture

Mostly based on [214], a list of state-of-the-art platforms for scientific agent-based simulation is detailed here below and compared with custom implementations. These can also be compared with the multi-robot simulators described in Table 2.3. Criteria of comparison include scalability and speed, ability to model complex systems, memory management, and the learning curve required to use such frameworks.

Swarm: implementing both a model for swarm hierarchies and a virtual laboratory, Swarm is based on schedules of actions that the objects execute. It uses its own data structures and memory management module. Initially implemented in Objective-C, it is a mature library platform, stable, and well organized. It lacks however some capabilities in error-handling and developer tools. Java Swarm is an attempt to access the Swarm Objective-C libraries from Java interfaces after a strong demand among users. Still, as a quick suboptimal

implementation, it seems to fail in combining the strengths of Java with the capabilities of the Swarm platform.

Repast: aimed at making the functionalities of Swarm available in Java, Repast is mainly focused on social science, hence lacking tools adapted for other domains. Despite improvements in interface and built-in examples, Repast has a steep learning curve.

MASON: designed with execution speed as a priority for complex models, it is less mature than the other alternatives but includes good parallelization capabilities.

NetLogo: originally an educational tool, it is focused on ease of use. It includes its own programming language but focuses mostly on one type of application: mobile agents interacting on a grid space with short local interactions.

ROS/Gazebo: the Robot Operating System (ROS) is an abstraction layer (middleware) used between simulated robots and real robots and it includes a simulator called Gazebo. This latter incorporates a powerful and fast physics engine able to simulate a wide range of sensors and apply textures to the environment. Based on a C++ implementation, this simulator is quite fast but would struggle with large number of robots in the group due to the high level of detail of the physics engine.

Additional tools available as resources at the Georgia Institute of Technology or at the Aerospace Systems Design Laboratory (ASDL) are also considered.

GUST: the Georgia Tech UAV Simulation Tool (GUST) includes tools developed for the aircraft of the Georgia Tech UAV Research Facility: onboard software, ground station, and simulation environment [215]. It also includes software-in-the-loop, a ground control station, sensor hardware, guidance computer, and primary flight computer testing capabilities. GUST is mainly designed to focus on the development and integration process before an aircraft leaves the laboratory. However, it remains mainly adapted to the type of platforms used by the Georgia Tech UAV Research Facility.

Robotarium: the project includes a remotely accessible swarm of robots as well as a simulator used to develop and test applications before trying them on the swarm. The aim of the project is to have a remote-access robotics lab where anyone is able to test their algorithms. The simulator is implemented in Matlab and is simple and fast enough for 2D simulations up to 100 robots.

NASA World Wind (modified by ASDL): developed by the National Aeronautics and Space Administration (NASA), this tool is adapted for displaying and interacting with geographic information. In particular, it can be used to implement a flight simulator and the ASDL has extended its capabilities in order to handle groups of robots.

As underlined by [214], these existing platforms should be improved in terms of statistical outputs and more importantly automating simulation experiments. Hence, in addition to these pre-existing platforms for agent-based modeling, the implementation of a custom model tailored to the needs of this research, is considered with three programming

languages. The first language being considered is MATLAB as it is widely used in the research community and includes many built-in libraries and functionalities. While MATLAB is easy to use and provides very good debugging and data visualization capabilities, it is significantly slow when compared to other programming languages. C++ is then considered as an alternative as its object-oriented language features would enable an easy implementation of an agent-based model. This language is quite fast but requires to precise the type of objects before using them. C++ also lacks built-in memory management tools, which requires the coder to manage memory carefully, an intimidating task to consider for agent-based modeling where many objects are involved. Java is a good alternative to C++ as it has proven similarly fast in recent years. Its error checking and garbage collection capabilities clearly facilitate the implementation of an agent-based model. Based on the previous descriptions, the capabilities of these platforms are summarized and compared in Table 3.7. The number of dots represent the degree to which a feature is present in the considered platform.

Table 3.7: Agent-based simulation platforms comparison

Simulator	Scalability & Rapidity	Complex Modeling Capabilities	Memory Management	Learning Curve
C++ based	●●●	●●●	●	●
Java-based	●●●	●●●	●●●	●
MASON	●●	●●●	●●	●●●
MATLAB (ASDL)	●	●●●	●●	●
NASA World Wind (ASDL)	●●●	●●●	●●●	●●●
GUST (GT)	●	●●	●●	●●●
Robotarium (GT)	●●	●●●	●●●	●
ROS/Gazebo	●	●●●	●●●	●
NetLogo	●●	●	●●	●●
Repast	●●●	●●	●●	●●●
Swarm (Objective-C)	●●	●●●	●	●●●
Swarm (Java)	●●	●●●	●●	●●

Based on the previous observations, the modified NASA ASDL model seems quite suitable for the considered problematics and is available as a resource. However, this model was added on the consequent NASA World Wind development kit and encompasses many dozens of classes and hundreds of methods adding a steep learning curve and quite some complexity compared to the required capabilities. Moreover, the architecture is limited in flexibility, making it hard to add new functionalities to the platform. NetLogo is also a very good option but the whole model code has to be written in one file, a choice which might

not be adapted to the complexity of this research. A Java-based implementation seems to regroup the benefits of both these options while enabling to keep a complex code organized. Nevertheless, the development would have to start a completely new implementation, adding a considerable lead time before starting any experiments. Hence, a reasonable learning curve is chosen as a main decision criterion so that the ROS/Gazebo and the Robotarium simulators can be adapted for respectively 3D microscopic and 2D mesoscopic models.

3.3.4 Testbed mission

Owing to the lack of standards of swarm engineering, no established canonical mission exists to test multi-robot systems. Hence, while this thesis proposes a generic framework for the design of multi-robot systems, a specific testbed mission has to be chosen to demonstrate its capabilities. In theory, any type of mission can be plugged in the framework and adapted to the requirements imposed on the designers. For this specific implementation, a large-scale topographical survey is chosen based on aerial imagery (Figure 3.15).

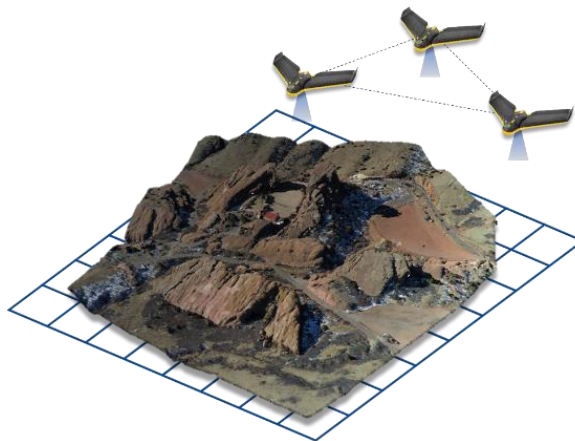


Figure 3.15: Representation of the testbed mission

[216] predicts that the agriculture drone sector will generate 100,000 jobs in the U.S. and \$82 billion in economic activity, clearly establishing the potential growth of such applications [217]. Leader in the field, SkyPlan advertises a coverage of 60 to 600 ha per flight using SenseFly drones [218]. As a reference, a soccer field has an area of 0.714 ha. Without cooperation, [219] shows that it requires 40 flights accounting for 22 total hours, in order to survey an area of 39 km². This requires 10 workers and 6 weeks are necessary to deliver the data to the customer. Such performance could greatly benefit from the use of multi-robot systems.

In addition, this type of mission proposes several advantages: as an area grid-based mission, it can be easily extended to other similar missions (mapping, surveillance, search and rescue, etc.). Such missions constitute a large component of the application spectrum for multi-robot systems. As such, it is a direct extension of the imaging/mapping canonical example given in section 2.1.

The implementation of such a mission relies mostly on a proper navigation scheme for the fleet depending on the terrain conditions, and the analysis used to stitch the captured data together. Strategies of exploration and mapping can be inspired from [211] and [220]. The following paragraphs detail the different steps required in the mission.

Aerial imagery: geotagged RGB images are captured from the drone (Figure 3.16) with typical high resolution cameras (around 16-Megapixel). The altitude of the drone determines the Ground Sampling Distance (GSD).



Figure 3.16: Aerial imagery [218]

A given GSD requirement hence limits the area covered in one flight. The quality required for such images implies some limitations on the flight conditions or on the performance of the system:

- No more than 50-degree flying angles
- Wind can affect data capture
- Low-hanging clouds can affect image quality
- Large water bodies or snow fields are unfavorable
- GPS signal can be erroneous between buildings

Orthomosaics: ortho-Mosaicking consists in the combination of two processes: ortho-rectification and mosaicking [221] (Figure 3.17).

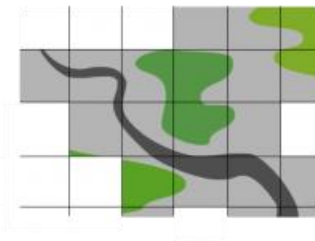


Figure 3.17: Orthomosaics [218]

The first one corrects the distortion and perspective effects due to the camera intrinsic parameters (focal length, optic center) and its angle with the vertical so that the map projection scale remains constant throughout the image. It may also account for relief effects to create an image which is plane and metrically correct. A visualization of common deformation effects is presented on Figure 3.18 and the process of ortho-rectification can be visualized on Figure 3.19.

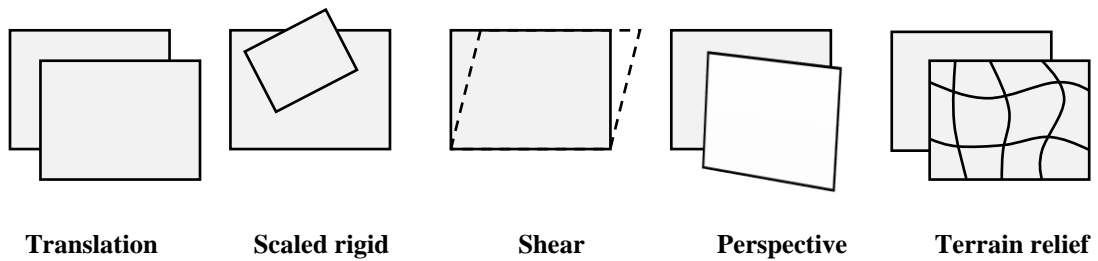


Figure 3.18: Common geometric transformations

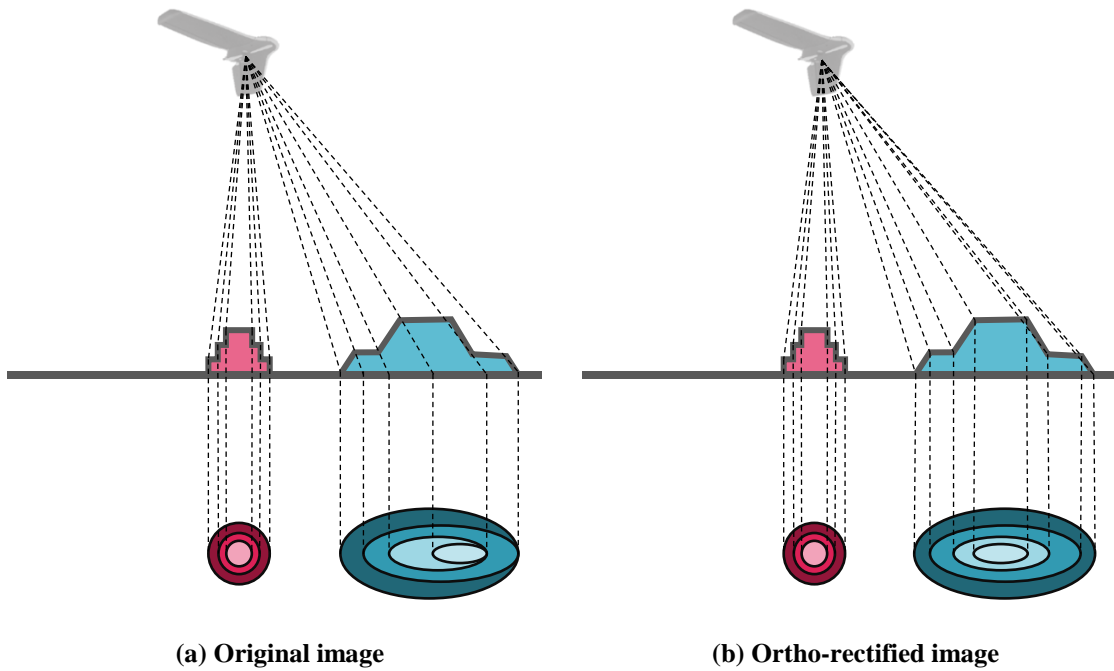


Figure 3.19: Ortho-rectification process

Finally, mosaicking is the process of stitching different images of the same scene together in order to obtain a single image (Figure 3.20).



Figure 3.20: Mosaicking [222]

The final orthomosaic obtained from this process enables the direct superposition of images with GPS data without any shift. The accuracy obtained on the final model is 1 to 2 times the GSD for horizontal coordinates and 2 to 3 times the GSD for vertical coordinates. Hence, a project with a GSD of 4 cm will most likely generate data with 3 to 6 cm of horizontal accuracy and 6 to 9 cm of vertical accuracy [218].

Surface models: thanks to the geotags available with each picture taken, it is possible to identify a very large number of matching features between the images and later generate 3D point clouds and surface models.

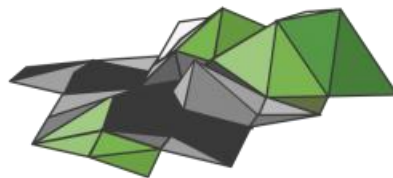


Figure 3.21: Surface models [218]

The sampling distance generally obtained for the surface models ranges from 2 to 4 times that of the captured images.

3.4 Decision-making process

Last step in the traditional top-down design procedure, the decision-making process aims at allowing designers to prioritize objectives and visualize tradeoffs between constraints and objectives. By using a priori optimization techniques, the optimization algorithm proposed in section 2.3.2 provides only one optimal solution. While this tends to suggest that there is no real decision-making step once optimization has been carried out, it relies on the assumption that the designer is able to correctly prioritize the objectives into the aggregate objective function. Hence, choosing the weights of the aggregate function is an indirect decision-making exercise as it affects the way each objective is considered to evaluate the performance of the multi-robot system. However, since the a priori knowledge of the effects of such weights on the optimal solution is not intuitive, it is difficult to qualify this as a true decision-making process.

A possible solution is to vary these weights in order to obtain several solutions and then compare them and make the decision process based on this set of solutions. By first varying the weights around the values chosen by the designer, it is possible to assess the robustness of the solution with respect to the requirements. It helps in making sure that a completely different solution is not obtained if the weights are slightly modified. This will ensure that the performance is the expected one once the system is implemented in reality with noisy sensors and manufacturing imperfections. Then, by giving completely different values to the weights, the designer is able to consider several weighting scenarios and chose

the best among them. Such a decision can be done using the Technique for Order Preference by Similarity to Ideal Solution (TOPSIS) method for instance [173].

3.5 Verification and validation

The step of verification and validation is essential to check on one hand that the proposed methodology is able to meet the requirements of the different experiments, and on the other hand that it correctly models multi-robot systems such as those used by the research community. The combination of both elements will ensure the consistency of the proposed approach. Note that third-party pieces of code which have already been verified and validated do not need to be included in this step.

First, the verification step is designed to ensure that the modeling part is done in line with the requirements established in the proposed approach. Its main purpose is to make sure that models are implemented correctly without any bugs or errors. For this matter, systematic unit tests are created and executed for the different algorithms at stake. Since most of the implementation is deterministic as per the proposed methodology, such unit tests are quite simple to implement and verify with synthesized data. Replications are used for stochastic processes in order to be able to draw consistent conclusions.

Then, the validation process certifies that the methodology is able to predict the performance of real-world systems. Given that swarm physical design optimization is a missing topic (see second chapter page 74), the validation of this part of the work is impossible by comparing results with existing systems. However, by enforcing the number of robots in the swarm to one, it is possible to compare the obtained swarm design with

existing single systems performing a given mission. This enables the validation of the design optimization scheme. Moreover, the modeling and simulation community is having a hard time finding an appropriate method to validate agent-based models [140]. Indeed, these are large complex systems which cannot easily be replicated in reality in the same way that simpler physical experiments can. A validation technique which can possibly be used in this particular case is to make sure that the comparison between different designs still holds when the detail and fidelity level of the models is varied. In particular, this helps in ensuring that a design remains optimal as the detail level is increased from conceptual level to actual implementation.

The implementation of the approach presented in this chapter, complemented with the solutions proposed in the previous one, provides a way to carry out the experiments designed earlier on to investigate the research objective. These experiments will then provide results possibly validating the hypotheses formulated in chapter 2 CHAPTER 2.

3.6 Summary

The problematic to be addressed has now been rigorously formulated and complemented with a proposed approach (Figure 3.22), this subsection summarizes the formal research process and the contributions anticipated from the research effort.

The design framework developed in this research is expected to bring advances in the exploration of extremely large design spaces with group architectures. Furthermore, the conceived methodology is also presumed to provide designers with a better understanding of the intrications and inner workings of multi-robotics. For want of a testbed application,

this research will also provide the field of multi-robotics with a testbed framework virtually extendible to any multi-robot application.

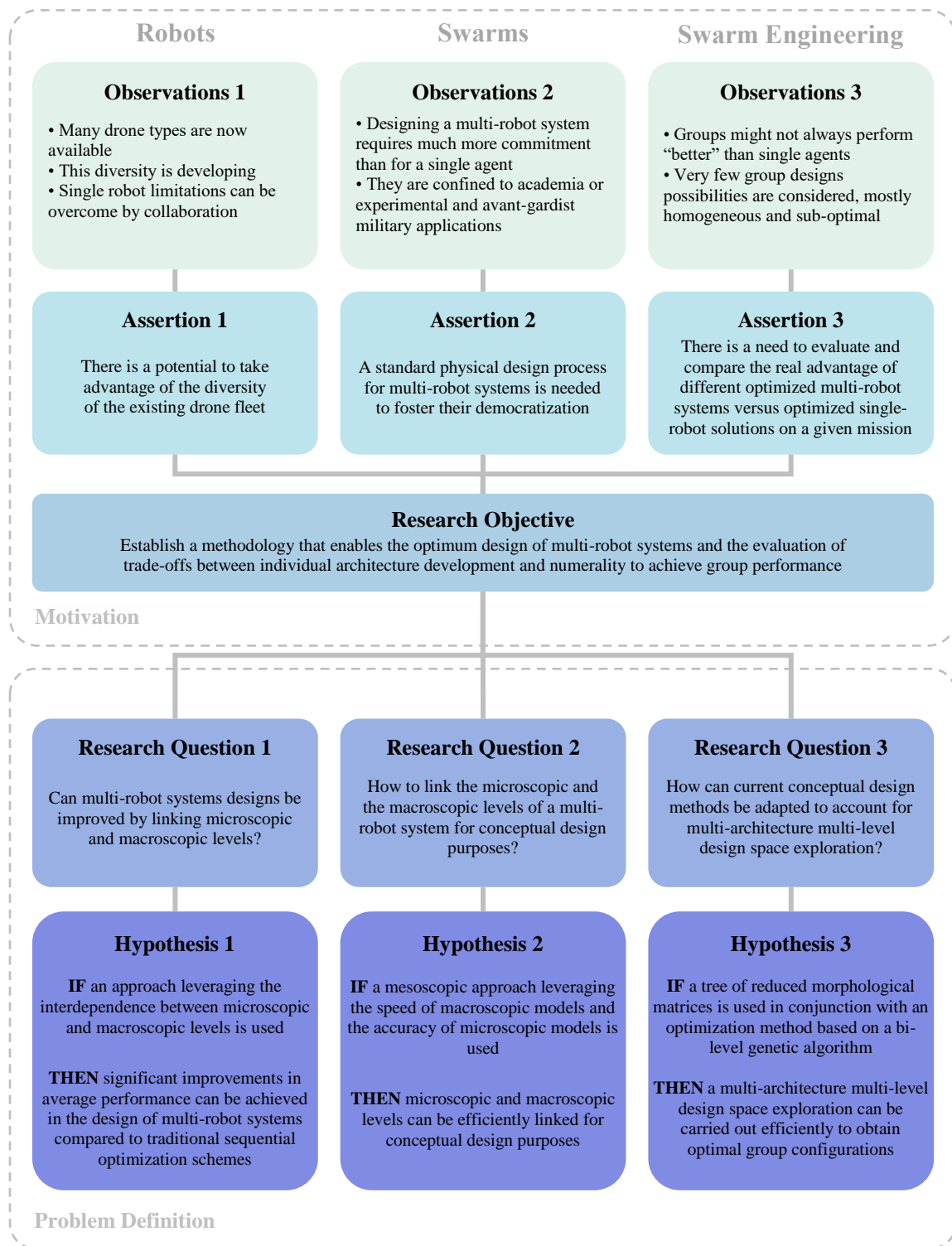


Figure 3.22: Summary of the research process

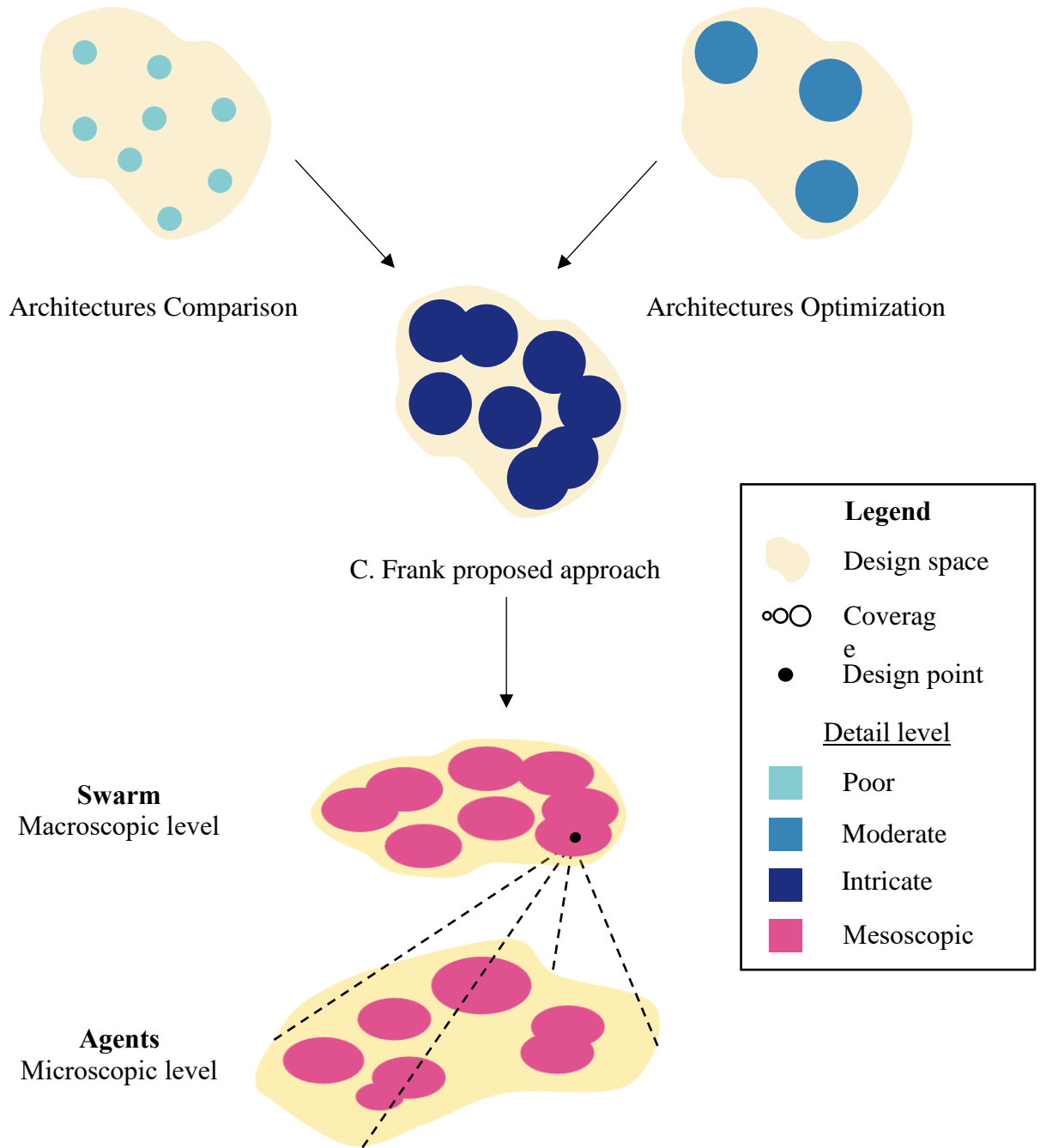


Figure 3.23: Proposed design space exploration technique summary

Design space exploration: the design domain for a swarm of robots often ends up involving extremely large numbers of designs to be studied, significantly due to numerality (Figure 1.37). Current approaches fail to properly study such design spaces in terms of both quantity of considered alternatives, but also quality of coverage. The presented design space exploration method (Figure 3.23) proposes a solution to explore extremely large design spaces which are generated when having to choose several designs defined among multiple architectures.

Insights on robotics swarm design: the present research originated from justified questions such as:

- Does using a swarm always provide increased performance for a given mission?
- After a change in mission requirements, is it better to increase the swarm size or increase the capabilities of each agent in order to still be able to complete the mission?
- Should a designer spend more time on developing the group architecture or on improving the R&D and individual performance of each agent?
- For a given mission, what is the optimal swarm architecture?

Answering this type of doubts, or rather providing designers with a framework to answer these questions, is part of the expected contributions of this work. The ability to elaborate and evaluate the performance of several robotic swarm architectures is expected to provide a new level of insight on their possible benefits. Not only will the performance be quantified with respect to standardized metrics, but it will also be compared with other

existing solutions on given requirements. This will provide answers to the need established in the third assertion of the motivation section: to evaluate the real advantage of swarms versus single-robot solutions on a given mission. For instance, the benefits of using a robotic swarm for imaging solutions (Figure 1.5) will be evaluated in terms of cost. Moreover, the performance losses incurred by sequential optimization of a swarm as opposed to the proposed methodology, linking microscopic and macroscopic level, will be measured and analyzed. This will provide indications on the type of performance improvements which could be achieved if an approach bridging the microscopic-macroscopic link was utilized by the community.

Finally, the introduction of performance metrics such as the marginal group performance is expected to provide precious additional insight on the benefits of adding entities to a group of robots. Used with different design variables, these marginal quantities predict the profile of the response and constitute a tool for design tradeoffs. On the other hand, the limit of parallelism effectiveness is an attempt to provide conceptual designers with an absolute reference of when the advantages of parallelization start to vanish.

Testbed framework: as mentioned in chapter 1, the design of multi-robot systems is particularly hard due to a lack of established simulators and standard testbed scenarios. If it seems impossible to find a universal benchmark application encompassing all the diverse aspects of multi-robotics such as collective exploration, chain formation, or coordinated motion; elaborating a standardized design methodology able to handle any application, is however achievable. Thanks to the proposed methodology and implementation, the testbed mission is completely modular depending on the actions encoded in each of the agents and

in the group logic. The individual behaviors have to be modified as well as the generic benchmarking metrics introduced in section 3.1.3. For example, the imaging testbed mission can easily be changed to a coordinated motion one. First, the individual tasks of mapping given areas have to be changed to tasks of forming and following the motion of neighboring agents. Then, the overall group strategy might also have to be changed at the swarm level. Once the mission is correctly implemented, appropriate metrics have to be used in the process in order to compare different swarm architectures for this given mission. For example, the mission performance metric can now be the time required to complete the desired motion and the mission completion metric can evaluate the quality of the swarm motion.

The next chapters detail the implementation of the experiments for each one of the main research axes (Figure 2.1 page 74), as well as the conclusions obtained from the results.

CHAPTER 4

LINKING MICROSCOPIC AND MACROSCOPIC LEVELS

Established as a main research axis in the previous sections, the lack of a link between the microscopic and macroscopic levels of a swarming system was decomposed into two sets of main research questions, hypotheses, and corresponding experiments. The first set focuses on evaluating whether a possible linkage between the two levels would incur improvements in the optimal design of multi-robot systems. Assuming that enhancements are indeed possible, the second set expands on how specifically this existing gap between the two levels could be bridged. This section details the experiments carried out and the results obtained in response to the first two research questions of this thesis work.

4.1 An improvement for the design of multi-robot systems

This first subsection details the study of the first research question and the first hypothesis, both of which are recalled here for reference.

Research question 1

Can multi-robot systems designs be improved by linking
microscopic and macroscopic levels?

Hypothesis 1

IF an approach leveraging the interdependence between microscopic and macroscopic levels is used
THEN significant improvements in average performance can be achieved in the design of multi-robot systems compared to traditional sequential optimization schemes.

In addition to the explanation provided in section 2.1, one may consider the macroscopic level and the microscopic level with one variable each only. It is then possible to easily visualize the problem posed by sequential optimization as opposed to global optimization. Indeed, looking at Figure 4.1, performing sequential optimization consists in optimizing based on X_{in} first and then X_{out} (micro-macro optimization), or vice-versa (macro-micro optimization). This translates into constraining the optimum search along a specific direction to reach a temporary optimum (circle), and then updating the search direction to reach a final optimum (star) from this previous temporary one. While this may work when the design variables are defined along the Eigen vectors of the fitness function (Figure 4.1), it fails when there is a correlation between the design variables (Figure 4.2).

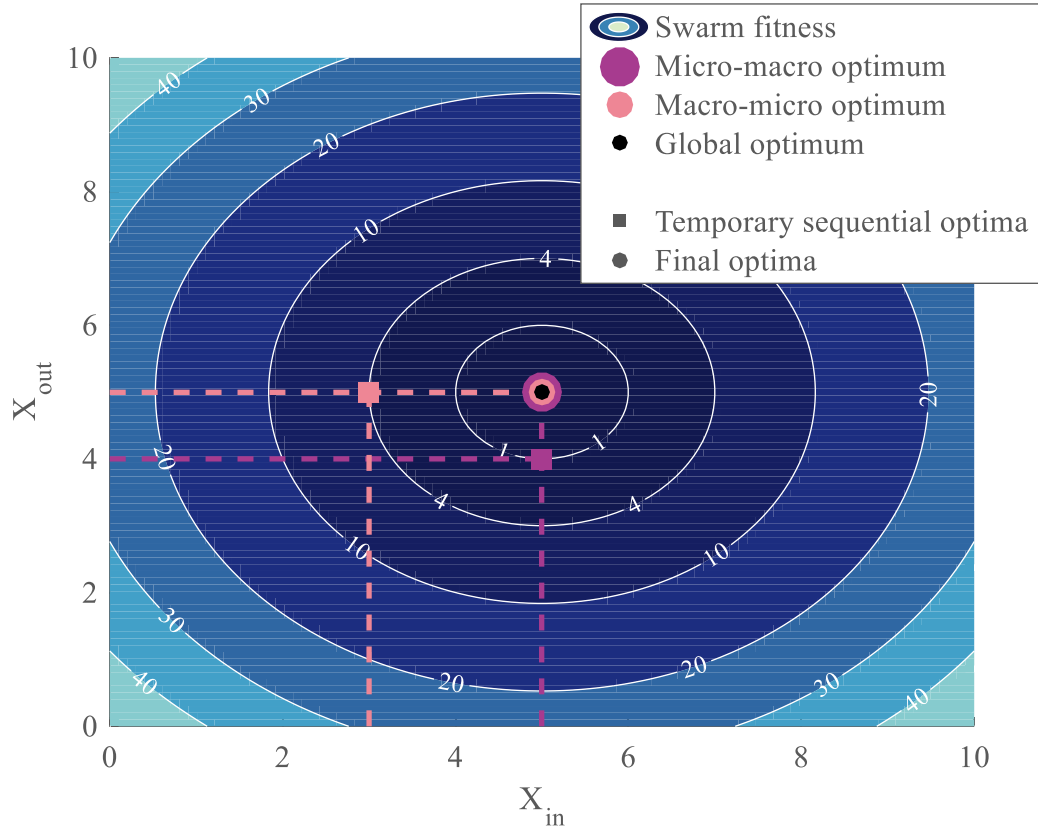


Figure 4.1: Sequential optimization without correlation

In this first figure, the fitness function for the swarm is modeled as:

Equation 4.1: Uncorrelated example function

$$f(X) = (X_{in} - 5)^2 + (X_{out} - 5)^2$$

The optimum search is started from an arbitrary value $X_{in} = 3$ for the macro-micro case and from $X_{out} = 4$ for the micro-macro case. Since there is no correlation term between X_{in} and X_{out} , sequential optimization is able to find the true optimum of the fitness function in both cases, independently of the starting values.

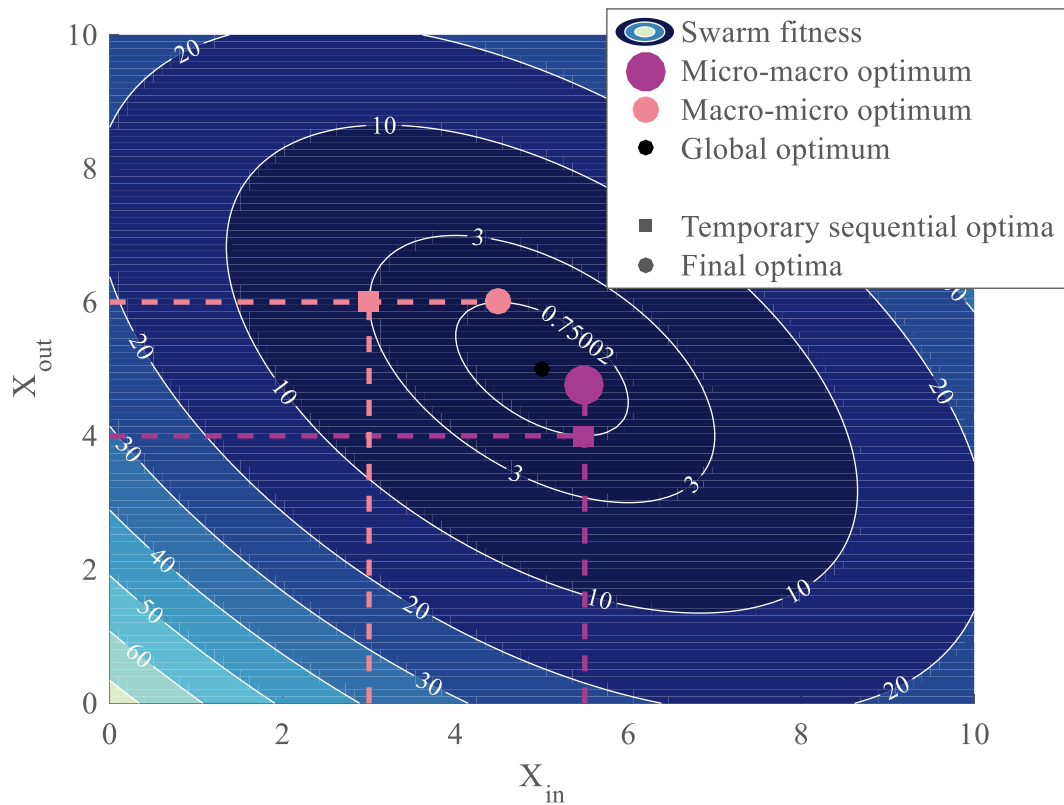


Figure 4.2: Sequential optimization with correlation

For the second figure, a correlation term is introduced:

Equation 4.2: Correlated example function

$$f(X) = (X_{in} - 5)^2 + (X_{out} - 5)^2 + (X_{in} - 5)(X_{out} - 5)$$

Note that the location of the global minimum is not affected. This time, starting from the same initial points as before, sequential optimization is not only unable to find the true optimum swarm configuration, but it also ends up in two different optima depending on the type of algorithm used. These local optima also depend on the initial points.

The goal of this section is then to answer the research question by concluding whether this type of observations can be made for multi-robot systems performance. The following subsections provide the key elements necessary for carrying out experiment 1 which compares sequential and global optimization in the context of multi-robot systems optimal design.

4.1.1 Global optimization algorithm

Key requirement of this first experiment, the role of the global optimization algorithm is to optimize simultaneously the macroscopic level as well as the microscopic level without separating them and having to optimize them sequentially. As explained in the hypothesis of section 2.3.2.2, a bi-level genetic algorithm is considered for this particular task. The reader should refer to section 5.2 for a complete analysis of this global optimization algorithm.

4.1.2 Sequential optimization algorithm

This first experiment also requires a sequential optimization algorithm to be compared to the global optimization one. In particular, both algorithms should reflect the same complexity so that they are somehow comparable in their precision and convergence time. Given that a genetic algorithm is chosen for the implementation of the global optimization algorithm (see section 5.2 page 345), the same choice is made for the macroscopic optimizer and the microscopic optimizer. Hence, the validated Matlab genetic algorithm solver is chosen to be the optimizer for the sequential optimization algorithm. As for the implementation, two cases have to be considered and are explained hereafter.

Micro-macro optimization: the agents are first optimized individually with respect to the mission requirements before the optimization of the group constitution X_{out} with respect to the same mission requirements (see Figure 4.3).

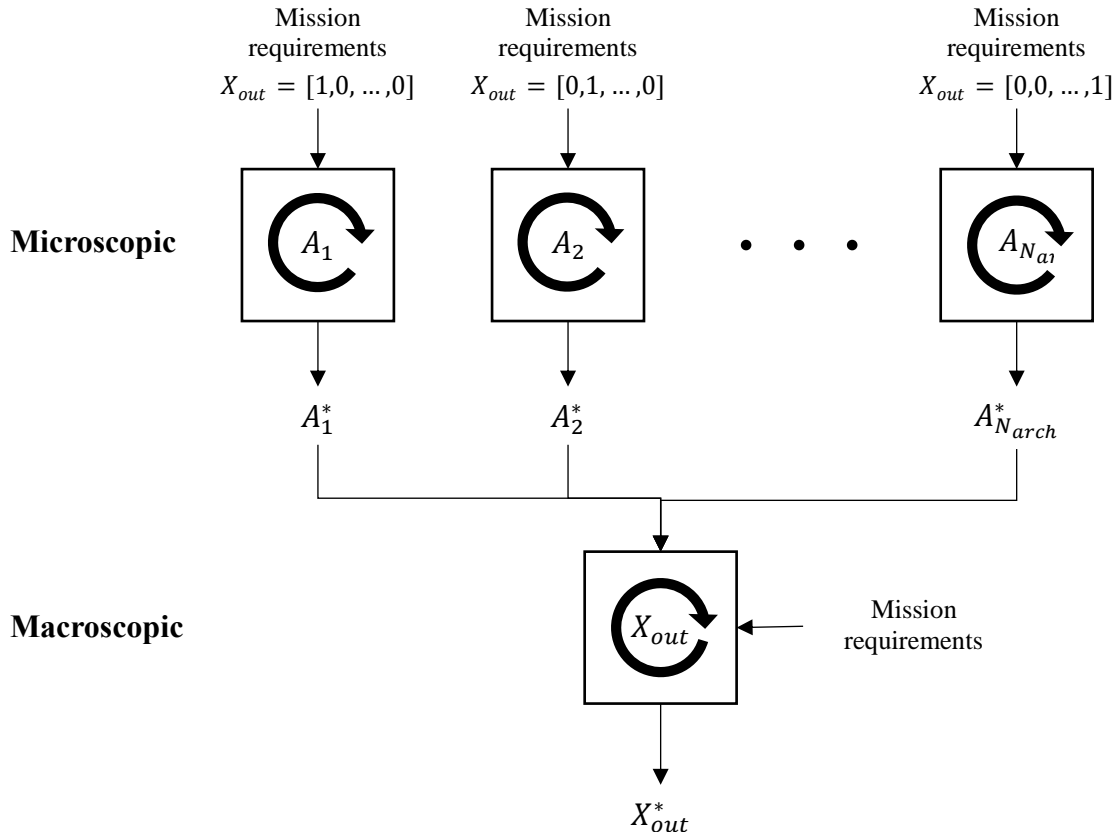


Figure 4.3: Micro-macro optimizer

Hence, for each architecture, one vehicle is set to perform the mission alone according to the mission requirements. This step yields individual vehicles which are each optimized to carry out the given mission alone. Then, the macroscopic optimizer handles the composition of the group as well as other macroscopic variables to decide how many optimal vehicles of each architecture will constitute the cooperative swarm. Each iteration sees the group perform the mission with the optimal vehicles obtained from the

microscopic optimization. This mostly corresponds to what is done in the current state of multi-robotics research: individual platforms are independently optimized for different missions and by different organizations, and put together to collaborate and carry out another mission. Note that here the vehicles are already optimized with respect to the same mission they will have to carry out in a group, which is generally not the case in reality.

Macro-micro optimization: the group composition is first optimized with respect to the mission requirements before optimizing the parameters of each constituent of the group with respect to the same mission requirements (see Figure 4.4).

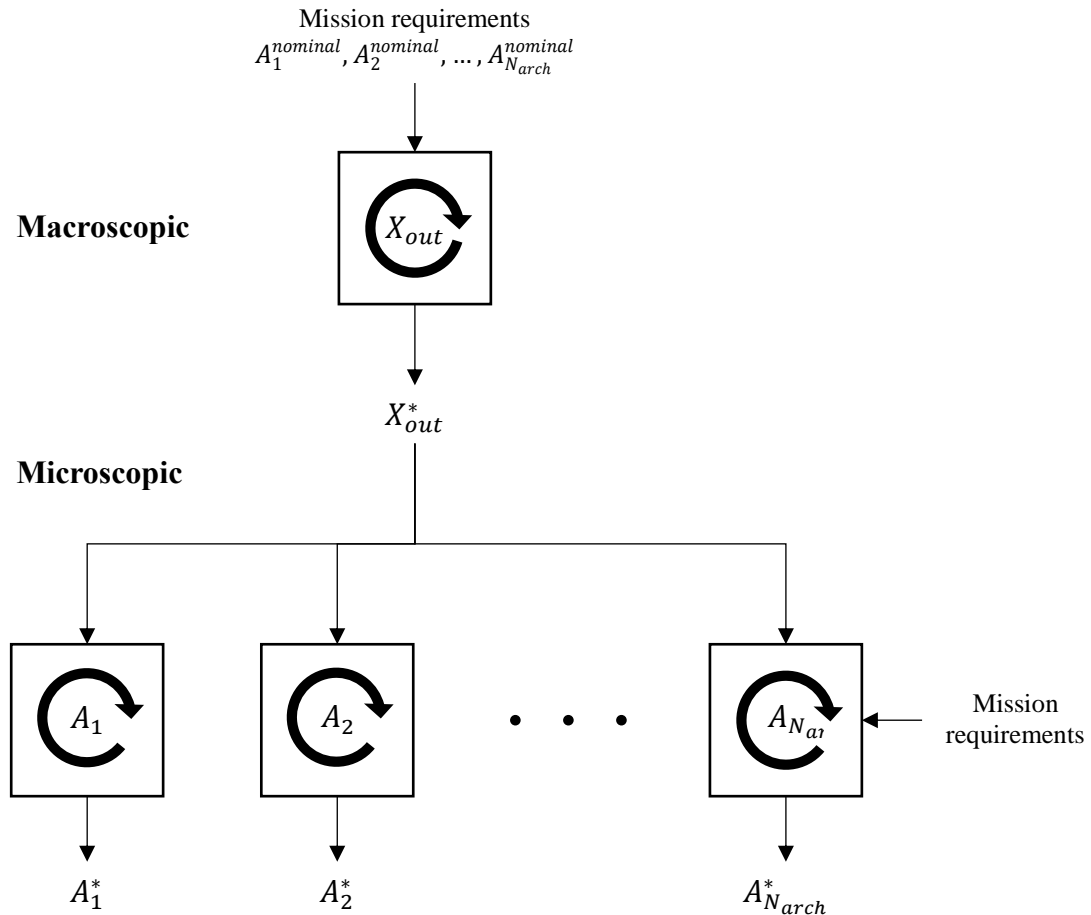


Figure 4.4: Macro-micro optimizer

This type of optimization is more delicate to carry out since a pre-existent baseline has to be available for each architecture. Considering these remarks, macro-micro optimization is almost equivalent to having an additional microscopic optimization after a micro-macro optimization.

It is important to note here that the notation used X_{out} and X_{in} is slightly different than the one used in the remainder of this thesis. In particular, when considering the sequential optimization scheme, X_{out} refers more precisely to X_{macro} while X_{in} represents in reality X_{micro} given the separation of microscopic and macroscopic levels when using sequential optimization.

In addition, in the case of the micro-macro optimization, the individual constituents of the swarm are optimized with respect to the same mission requirements that the group will face. This is generally not the case for what is currently observed in the research community: scientists tend to gather platforms which designs were optimized for missions which are different from what the swarm is required to carry out. For instance, research groups might purchase DJI Phantom drones which are optimized for aerial photography, and make several of them work collaboratively on a formation flight application with different requirements. Subsequently, it is expected that the results obtained throughout this experiment will be pessimistic with respect to the actual improvements that could be obtained in the optimal design of multi-robot systems.

4.1.3 Verification and validation

The macroscopic optimizer and the microscopic optimizer are both based on the Matlab genetic algorithm which is a validated piece of software. Hence, there is no need to further validate these two loops. However, the validation of the global optimization algorithm is required and detailed in section 5.2.2 page 364. Moreover, the fitness functions (simulation models) used in conjunction with these algorithms are validated in their respective chapter.

4.1.4 Experimentation

The experiment used to study research question 1 and demonstrate the interest of simultaneous optimization for multi-robot systems is the same used for the introductory example of section 2.1. The reader can hence refer to page 75 for a complete description of the canonical mission and the derivation of the corresponding macroscopic model. This simple model is sufficient and adequate to answer the first research question as it contains all the elements required to establish a surrogate of multi-robot missions. Indeed, it contains mobile robots with individual capabilities in terms of sensors and motion. Moreover, group dynamics are emulated through a particular deployment scheme that involves different traveling distances for each of the agents of the group. This canonical mission is used in the micro-macro, macro-micro, and simultaneous optimizers in the following subsections for conclusions to be drawn.

Note that in the case of micro-macro optimization the architectures are optimized as if they had only one vehicle performing the mission. Once these will be integrated into a swarming system, all the vehicles for a given architecture will hence have the same

configuration: this is the same as considering only a partial heterogeneity (see section 5.2.1 and Figure 5.35 for details). As a consequence, no additional experimentation is done in this section to study the effects of full heterogeneity versus partial heterogeneity in the group since partial heterogeneity is intrinsically enforced.

4.1.4.1 Homogeneous swarms

4.1.4.1.1 First strategy: optimizing a single vehicle

The micro-macro and global optimization on the canonical mission were already performed a first time in section 2.1 using a full factorial design of experiments on the considered design space: $N \in \llbracket 1, 10 \rrbracket$ and $v \in [0, 10]$ m/s with steps of 0.1 m/s. This time, the different optimizers presented in the previous subsection are used to draw conclusions and the macro-micro optimizer is also introduced. For each of the cases presented hereafter, the three optimizers are run and compared with respect to the criteria established for experiment 1.

Experiment 1.1: First, the mission and cost parameters are fixed as follow: $d_0 = 100$ m, map size of $l_x = 100$ m by $l_y = 100$ m, agent fixed cost $c_0 = 3$, swarm fixed cost $C_0 = 10$ and unit cost of individual performance (velocity) $c_v = 1$ (m/s)⁻¹. The cost constraint is fixed at $C_{max} = 70$. The baseline velocity used for the macro-micro optimization is fixed at 5 m/s which corresponds to the average of the bounds of the design space. Figure 4.5, shows the results of this first experiment. The three optimum solutions are represented with the cost-constrained region. In addition, the second search directions used by the sequential algorithms are represented. For instance, for the micro-macro optimized, the search direction is along the optimal velocity found by the micro optimization.

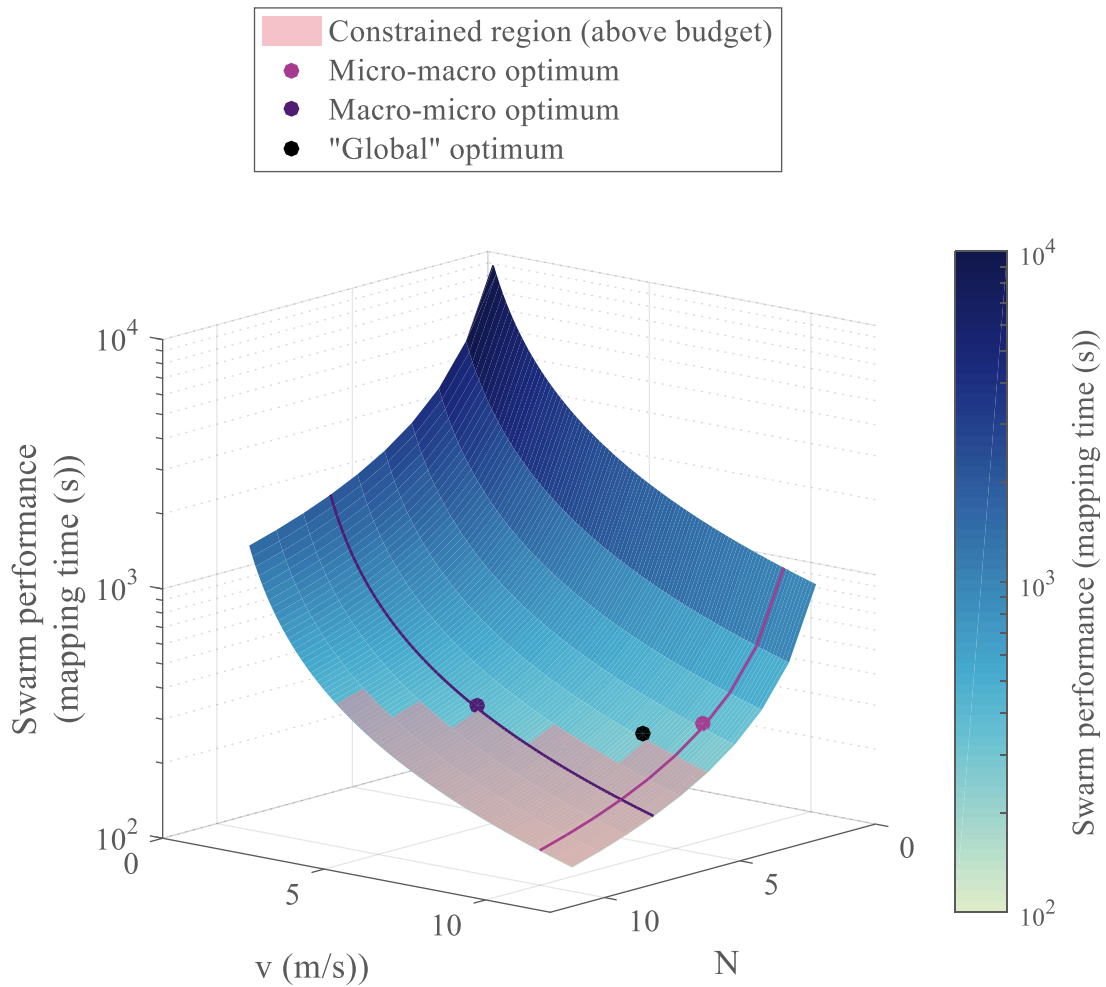


Figure 4.5: Plot of experiment 1.1

Similar to the results found in the canonical example, the micro-macro optimal swarm composition is found to be 4 vehicles at 10 m/s each for a total cost of 62 units and a mission mapping time of 285 seconds. The whole set of results is presented in Table 4.1, with the relative difference with respect to the solution found by the simultaneous optimizer.

Table 4.1: Results of experiment 1.1

Variable/Metric	Macro-micro	Micro-macro	Simultaneous
N	7 (+ 40%)	4 (- 20%)	5
v (m/s)	5.57 (- 38%)	10 (11%)	9
Mapping time (s)	323 (+ 23%)	285 (+ 9%)	262
Cost	70 (0%)	62 (- 11%)	70

A first observation is that the solution derived by the macro-micro optimizer seems less performant than the other two although it reaches the maximum cost constraint. This can be explained in the following way: the macro-micro optimizer first decides on a number of vehicles to integrate in the swarm which still fits inside the cost constraint. Since the number of vehicles can be increased only one by one, the cost increases a lot for each additional vehicle. Once the number N of vehicles is decided, the micro optimization takes place and tries to maximize the velocity of the N agents so that the mapping time decreases. However, the velocity increments are much finer than the numerality increments and the optimizer is able to get much closer to the cost constraint. On the contrary, the micro-macro optimization finishes with the macro optimization, taking much larger increments in cost (by changing only the number of vehicles) and does not have such a refinement. Hence, the cost of the micro-macro optimization is far from the budget constraint of 70 cost units despite proving a better performance in mapping time.

Indeed, in this first micro-macro optimization, the micro optimization is constrained by the same cost constraint as for the macro optimization. This means that a single vehicle could cost up to 70 units. While this clearly illustrates one of the points of micro-macro optimization where more resources can be allocated to the microscopic level,

it does not correspond to real-world situations. Instead, the cost of a single agent should be decently constrained for the micro optimization, this is illustrated in the next micro-macro optimization.

Another observation is that the macro-micro optimizer favors numerality (point on the left of the global optimum) whereas the micro-macro focuses on the individual performance of the agents (point at the right of the graph).

Experiment 1.2: For this second experiment, the cost constraint on a single vehicle is fixed at 10 units of cost, a limit slightly lower than the maximum cost achievable: 13 units. Hence, the micro-macro optimizer can no longer opt to simply assign the maximum velocity for each agent and then assemble a group of these elite agents. This time, the micro-macro optimum has a much lower velocity than before: the cyan search direction has shifted to the left on Figure 4.6. With this new technique, the results are listed and compared in Table 4.2.

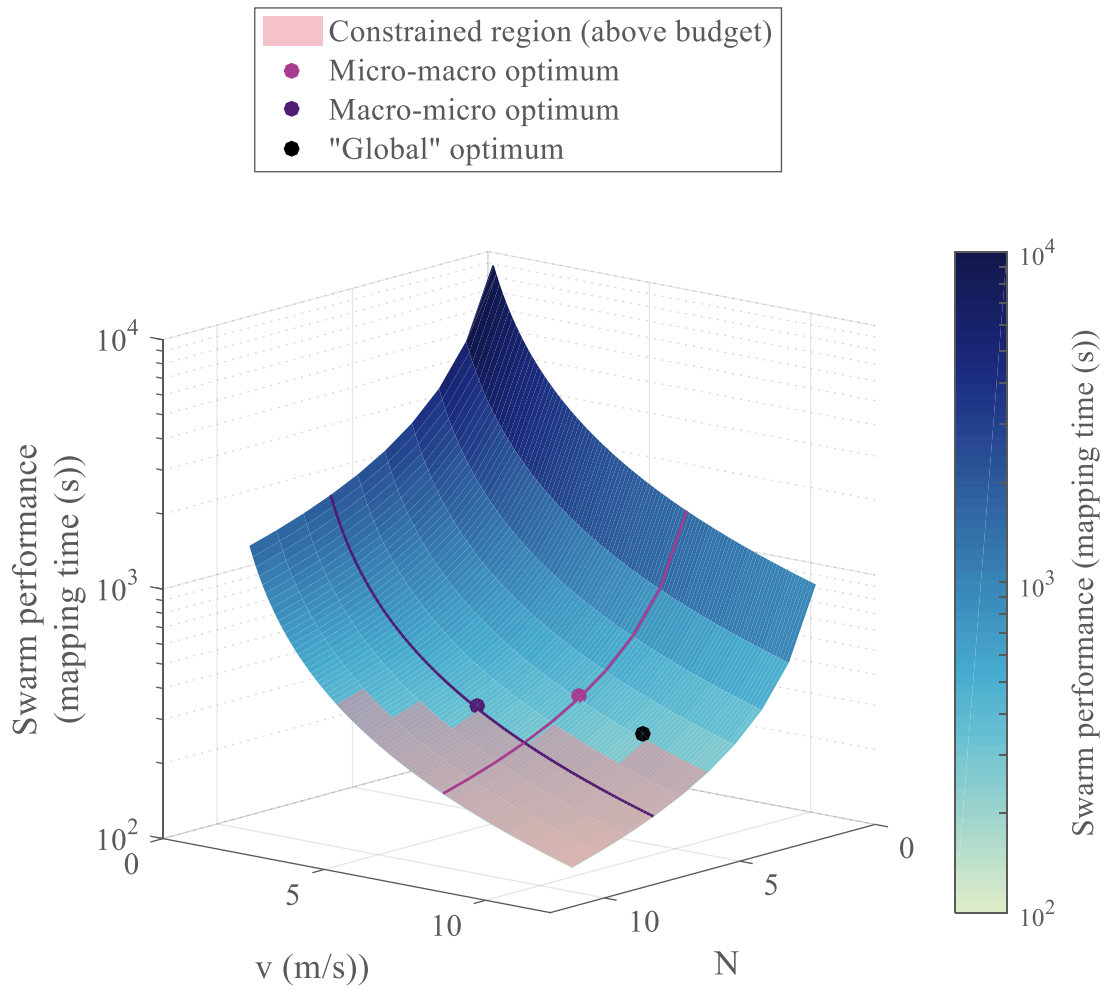


Figure 4.6: Plot of experiment 1.2

Table 4.2: Results of experiment 1.2

Variable/Metric	Macro-micro	Micro-macro	Simultaneous
N	7 (+40%)	5 (0%)	5
v (m/s)	5.57 (-38%)	7 (-22%)	9
Mapping time (s)	323 (+23%)	337 (+29%)	262
Cost	70 (0%)	60 (-14%)	70

As expected, the macro-micro optimization results are unchanged. Being more constrained for the micro optimization, the micro-macro optimizer now exhibits a worse performance than in the first experiment. In fact, it obtains the same number of vehicles as the simultaneous optimization but since it is sequential, it is not able to dynamically derive a constraint on the vehicle cost to obtain a cost closer to the constraint and hence maximize the performance of the swarm. This time, the performance difference in mapping time is worse by 23 to 29 % for the sequential optimizers.

Experiment 1.3: As a third experiment, the cost of individual technology is increased thanks to a quadratic term $c_{v_2} = 0.3$ m/s and the total cost constraint is increased to 100 units. The goal is hence to slightly favor numerality and give more freedom to the algorithms to compose a swarm and see in this case if the sequential algorithms are able to match better with the performance of the simultaneous optimizer. As in the first experiment, the micro step of the micro-macro optimizer is first constrained with the same cost constraint as the whole group, Figure 4.7 is obtained.

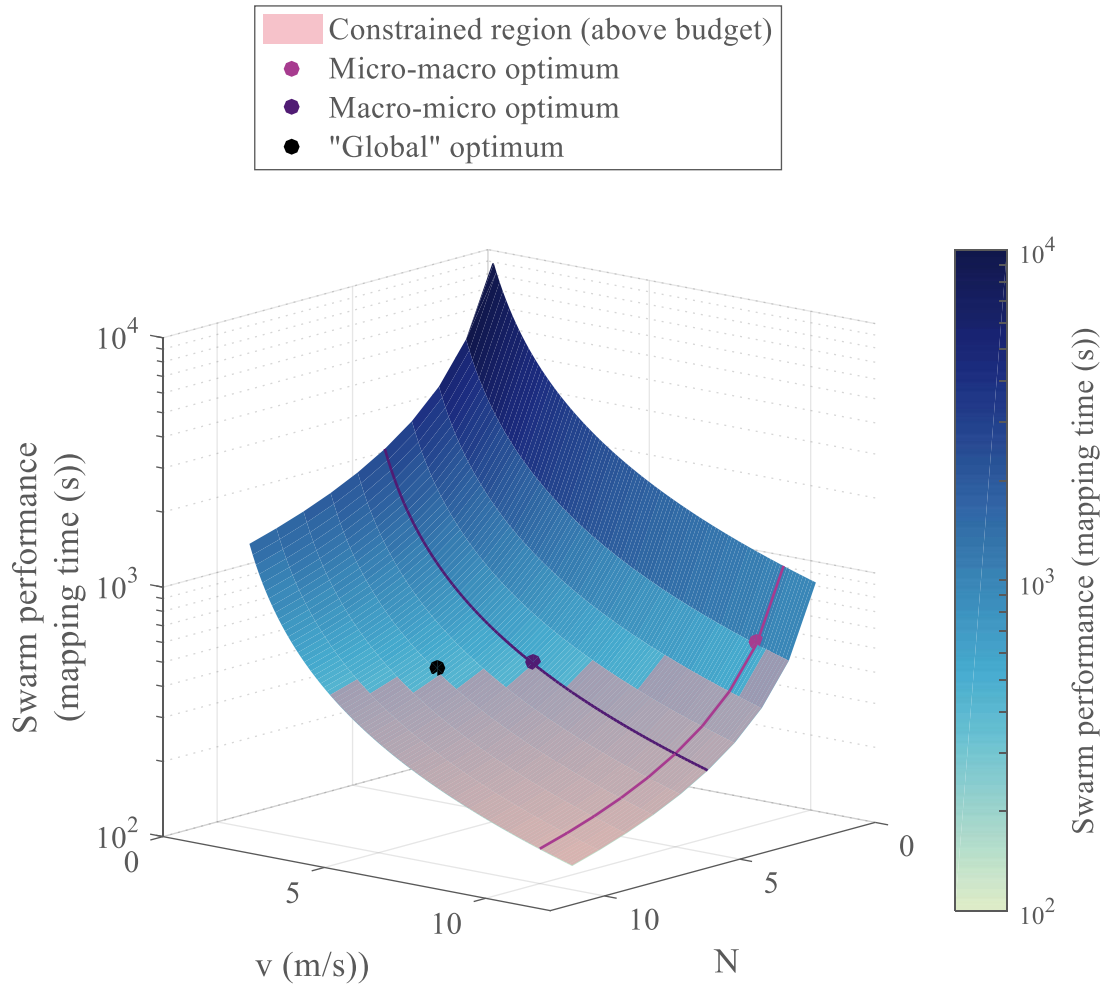


Figure 4.7: Plot of experiment 1.3

Table 4.3: Results of experiment 1.3

Variable/Metric	Macro-micro	Micro-macro	Simultaneous
N	5 (-29%)	2 (-71%)	7
v (m/s)	5.60 (+30%)	10 (+133%)	4.30
Mapping time (s)	422 (+1%)	530 (+27%)	418
Cost	100 (0%)	96 (-4%)	100

A first observation is that the micro-macro optimizer is able to get much closer to the cost constraint than it did in the first two experiments, hence efficiently utilizing the assigned budget. However, once again, it favors very expensive individual vehicles with high performance and finds itself very limited for the macro step once it is time to assemble a swarm made out of these vehicles. As a consequence, only two vehicles constitute the optimal group derived by micro-macro optimizer. Despite being within 4% of the cost constraint, the performance is 27% slower than the optimum swarm derived by the global algorithm. On the other hand, the macro-micro optimum is quite close (within 1% of performance) to that best optimum while utilizing the total budget.

Experiment 1.4: Finally, the cost of a single vehicle is then limited to 10 units as for the second experiment in order to get closer to real-world situations and force the micro-macro optimizer to favor numerosity (see Figure 4.8 and Table 4.4).

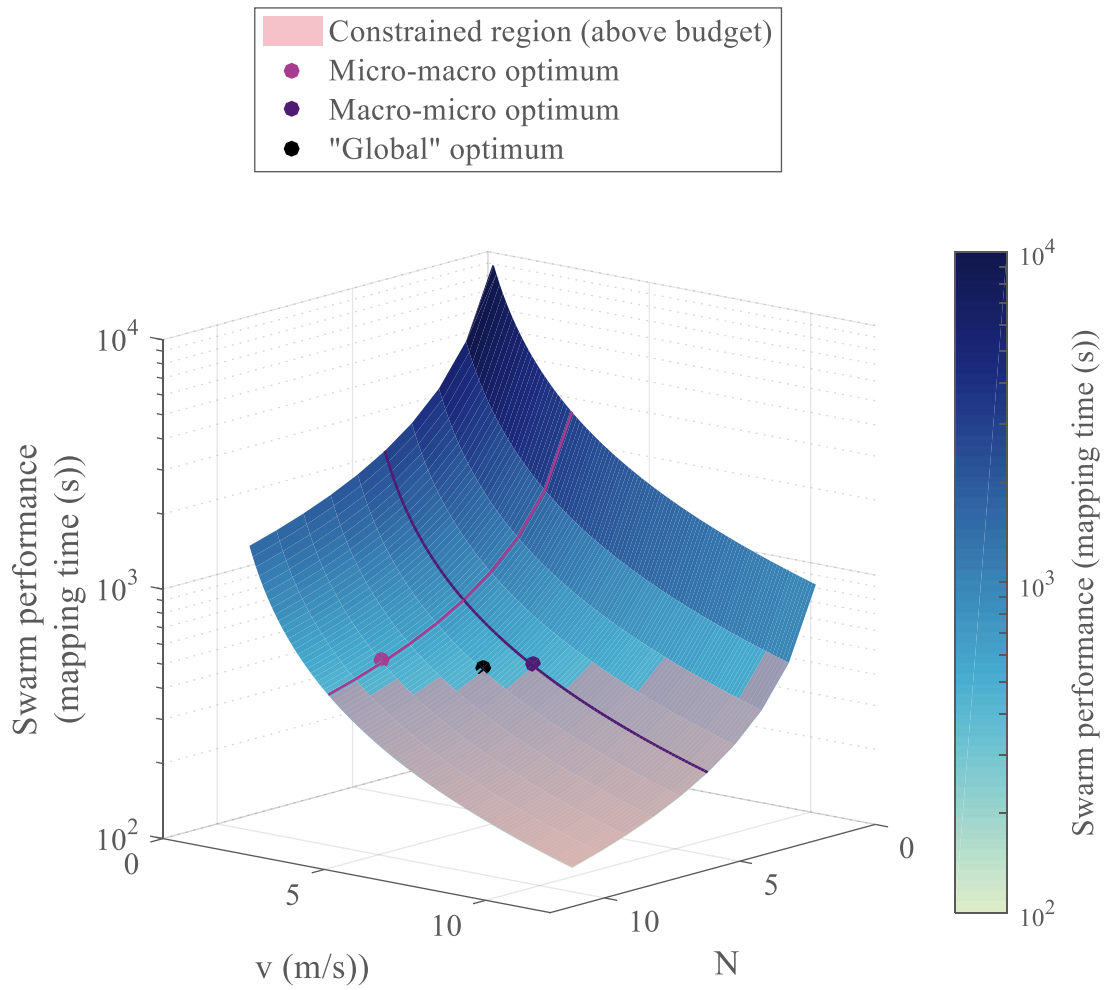


Figure 4.8: Plot of experiment 1.4

Table 4.4: Results of experiment 1.4

Variable/Metric	Macro-micro	Micro-macro	Simultaneous
N	5 (-17%)	8 (+33%)	6
v (m/s)	5.60 (15%)	3.44 (-29%)	4.87
Mapping time (s)	422 (+1%)	472 (+13%)	417
Cost	100 (0%)	90 (-10%)	100

Again, the micro-macro optimizer chooses a velocity as close as possible to the individual cost constraint which is 3.44 m/s in this case. It is hence much more conservative on the technology of the individual agents due to the additional quadratic cost term. This forces it to focus more on numerality and the difference is flagrant with the previous cases: the micro-macro optimizer now chooses 8 vehicles (for a maximum of 10) and is able to obtain a performance much closer to the best optimal one.

Additionally, the macro-micro optimizer also predicts a microscopic design for the individual agents which is very close to the one used for the macro optimization step. Indeed, the nominal agent velocity was fixed at 5 m/s for the macro optimizations and the algorithm ends up preferring velocities close to 5 m/s on the final design.

4.1.4.1.2 Second strategy: optimizing a notional swarm

Building on the results of the first four experiments, it seems that the micro-macro performs poorly when used with the same cost constraints as the group since it is designed to optimize vehicles individually. Indeed, the first step of the optimization is to make one single agent perform the mission and optimize its performance with respect to a given cost constraint. This is representative of the current habits of the research community: taking vehicles which were individually optimized for specific types of missions, and putting them to work together in a robotic-group. Nonetheless, to fully characterize the potential of micro-macro optimization, it is essential to compare it in equal terms with macro-micro optimization. Since this latter uses a vehicle baseline to first derive an optimal number of vehicles for the group, it makes sense for the micro-macro optimizer to use a swarm baseline (instead of one single vehicle) to derive an optimal individual agent velocity.

Hence, this part details the same experiments as the previous subsection by setting the number of vehicles to five for the micro optimization step: average of the bounds on the number of vehicles, similarly to what was done for the baseline velocity. However, given that the micro optimization (first step of the micro-macro optimization) is now performed on a reasonable size of swarm, there is no motivation to adjust the cost constraint for this first step anymore as it was done for experiment 1.2 and 1.4. As a consequence, experiments 1.5 and 1.6 respectively have the same inputs and requirements as experiments 1.1 and 1.3.

Experiment 1.5: Similar to experiment 1.1 with the new micro optimization requirements, this experiment shows that the micro-macro optimizer is now able to match exactly the performance of the simultaneous optimizer. Indeed, the optimal number of vehicles found by this latter was 5: the exact same number used for the baseline swarm used in the first step of the micro-macro optimizer. Hence, the second step tries to optimize the individual velocities for a swarm of 5 agents and ends up finding the same velocity as the global optimizer: 9 m/s. The results obtained by experiment 1.5 are then similar to experiment 1.1 with the micro-macro optimal design matching the simultaneous optimal design (see Figure 4.9 and Table 4.1 page 243).

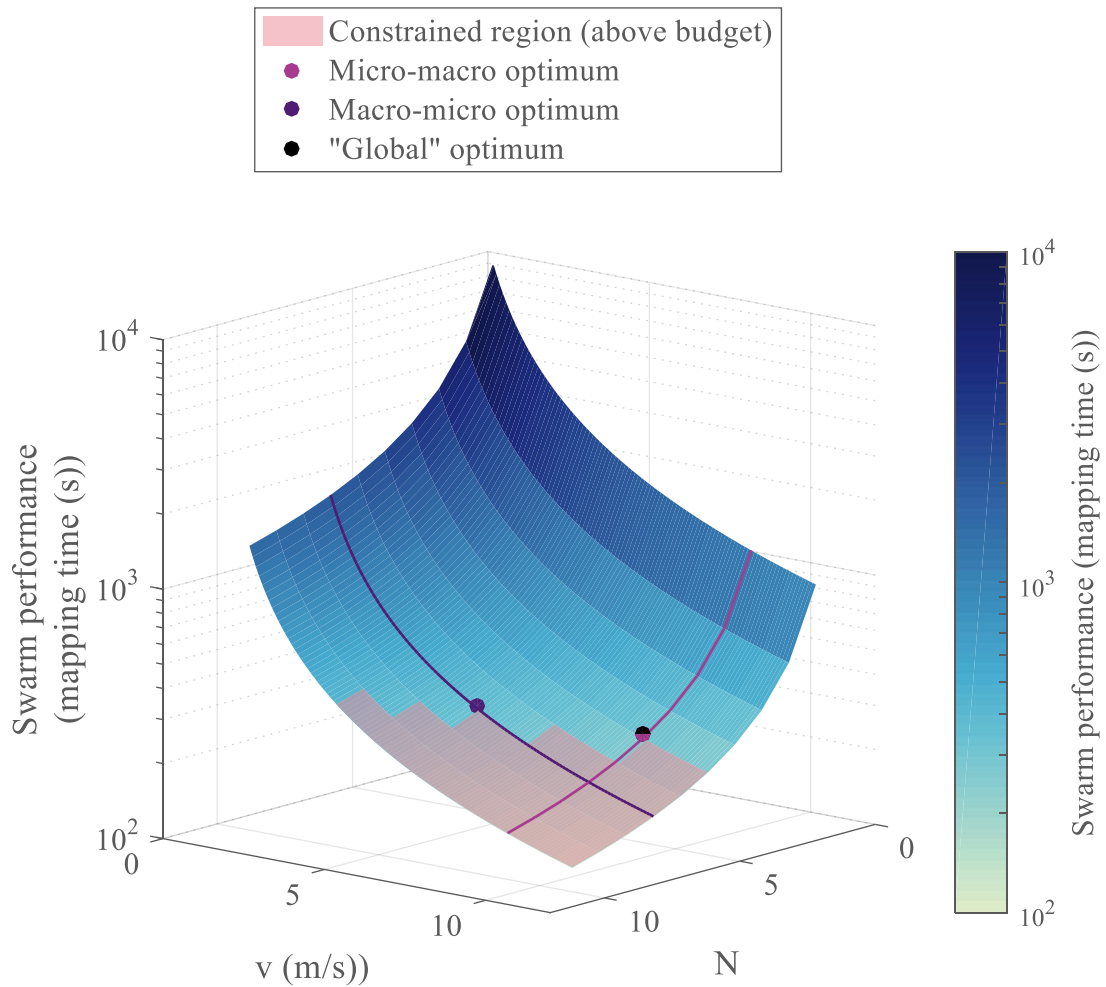


Figure 4.9: Plot of experiment 1.5

Experiment 1.6: Identical to experiment 1.3 with the new micro-macro optimizer, experiment 1.6 exhibits another particular case where the micro-macro and macro-micro algorithms derive the same optimum (see Figure 4.10). Once again, this is due to the fact that the number of vehicles used for the baseline group is 5, the same number derived by the macro-micro optimization. This can be seen on the plot as well since the optimal points are more or less centered in the design space. This result is purely aleatory here and depends on the structure of the problem, this observation cannot be generalized.

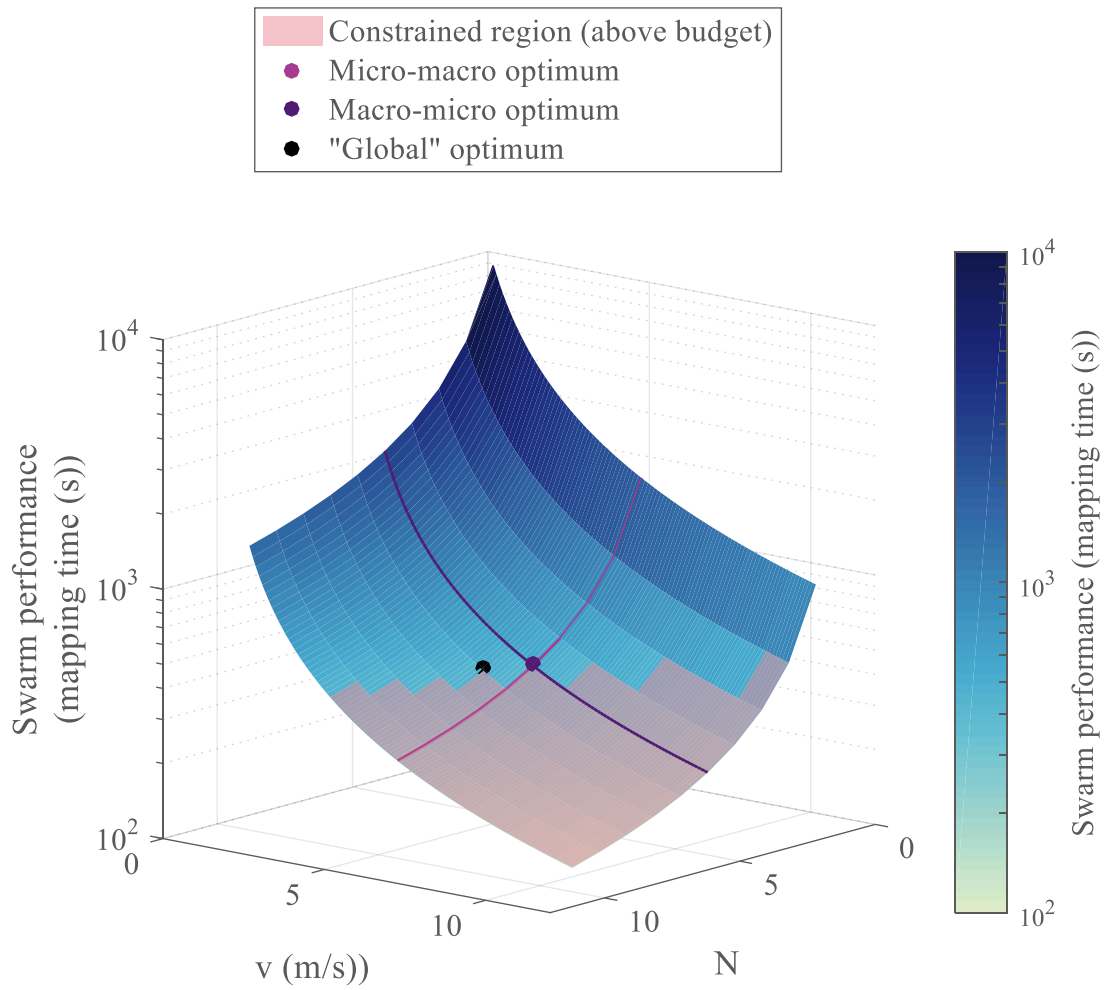


Figure 4.10: Plot of experiment 1.6

Table 4.5: Results of experiment 1.6

Variable/Metric	Macro-micro	Micro-macro	Simultaneous
N	5 (-17%)	5 (-17%)	6
v (m/s)	5.60 (15%)	5.60 (15%)	4.87
Mapping time (s)	422 (+1%)	422 (+1%)	417
Cost	100 (0%)	100 (0%)	100

Analogy with coordinate search: thanks to the different representations of the previous experiments, one quickly understands that, in this particular case, the micro-macro and macro-micro optimization algorithms correspond to search directions for a minimum along the coordinate directions N and v . However, these coordinate directions are not aligned with the eigenvectors of the response, hence yielding suboptimal results due to the conditioning of the problem [195]. This is illustrated on Figure 4.11 here below with an experiment similar to experiment 1.4.

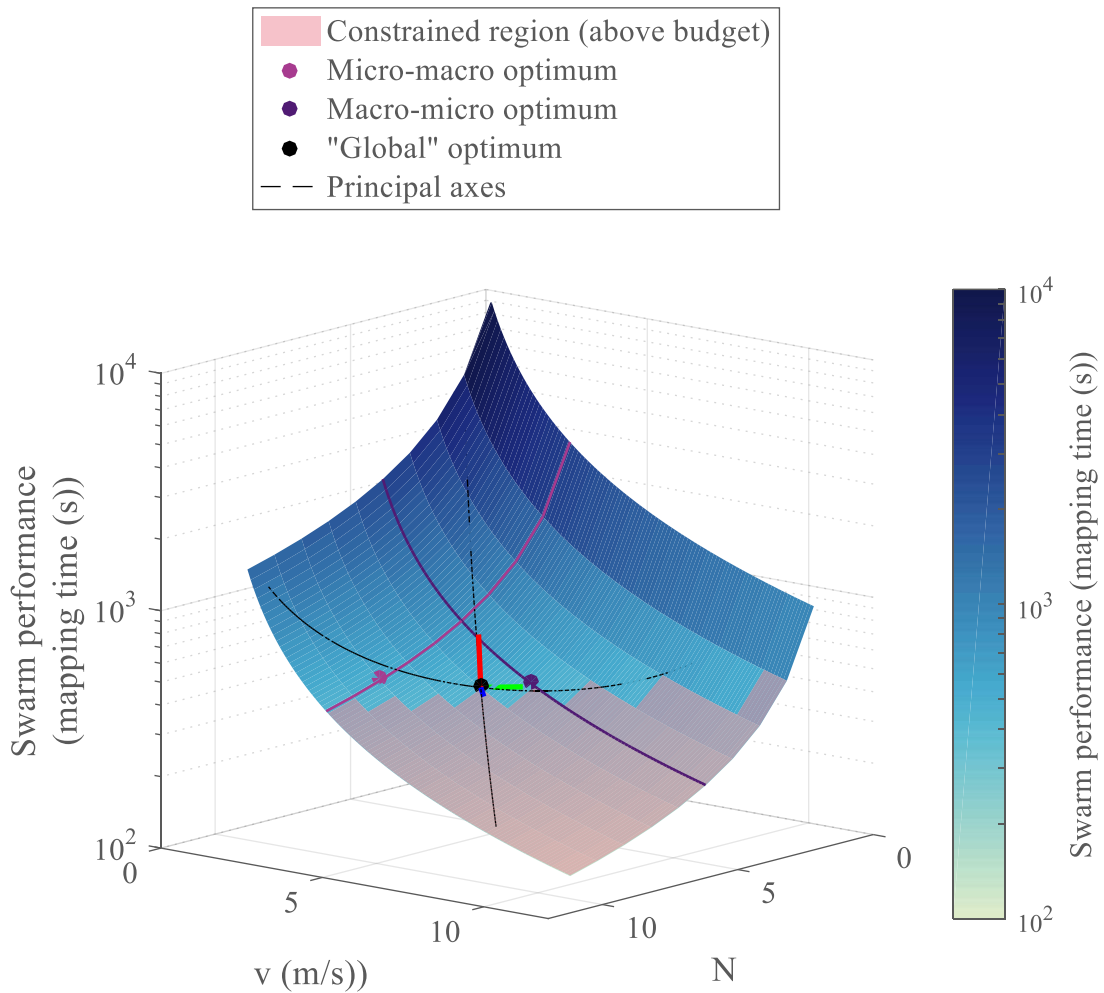


Figure 4.11: 3D visualization of principal component analysis

The principal basis of the response function is represented by the red-green-blue (RGB) triplet at the global optimum. The principal directions are then represented by the black dashed lines. As it can be seen, these do not align with the coordinate axes and the cyan and magenta search directions of the sequential algorithms. A similar projected representation is proposed on Figure 4.12. For visualization purposes, the constraint has been plotted as a continuous contour of the 70 cost level as opposed to the previous plots where the discrete aspect of the number of agents is emphasized. Note that the cyan micro-macro optimum is not even able to come close to the cost constraint and is under-utilizing the available budget.

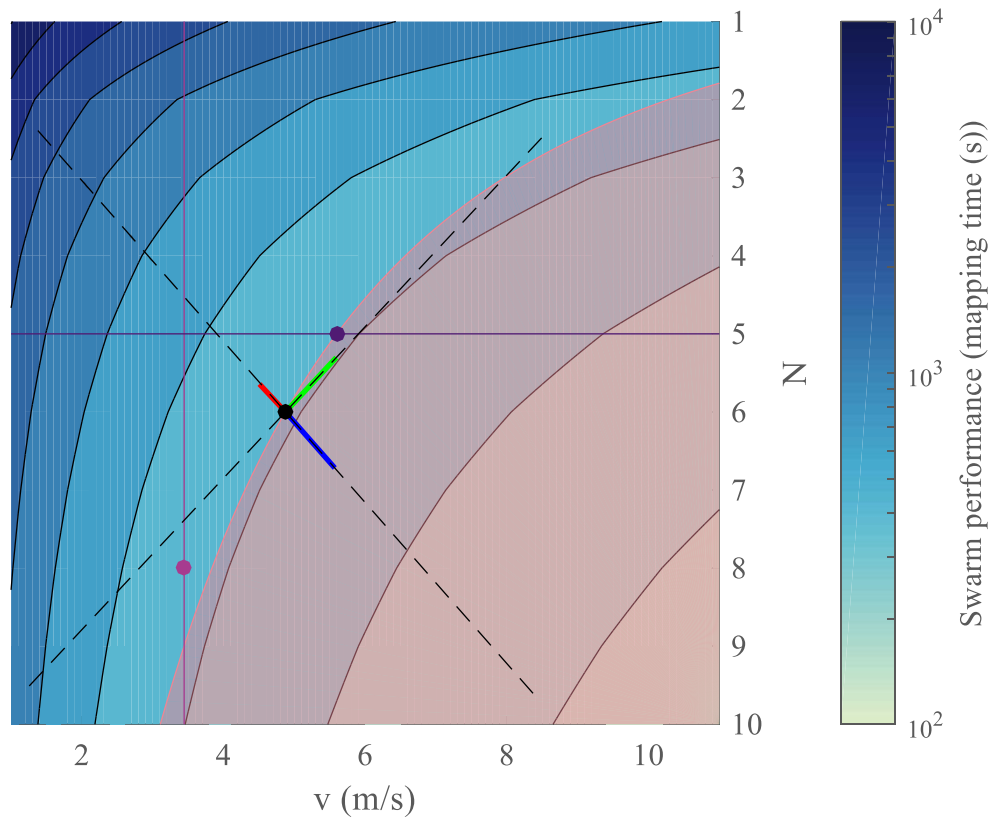


Figure 4.12: 2D representation of principal component analysis

Strong from this analogy with a coordinate search optimization algorithm, it seems that the chances of sequential algorithms leading to optimal results with multi-robot applications are extremely unlikely except for exceptional particular cases as it was shown in experiments 1.5 and 1.6.

4.1.4.2 *Heterogeneous swarms*

A relative heterogeneity can be added to the canonical example by considering groups of robots exploring different areas of the map (Figure 4.13). For instance, faster units can be deployed at the very end of the map while the slower agents stay at the proximity of the deployment point, hence taking advantage in the heterogeneity of the group to finish the mission in a reduced time. The formulation is quite similar to the homogeneous one except that now, the performance of the whole system is dictated by the slowest agent in all the groups. The assumption is that each robot takes the same amount of time to map their assigned area. Hence the faster robots will be assigned larger areas to map than the slower ones. Note that the slowest agent of a given robot group is, as before, the agent being deployed the furthest. Using this approach, the effect of heterogeneity on the group performance can be analyzed.

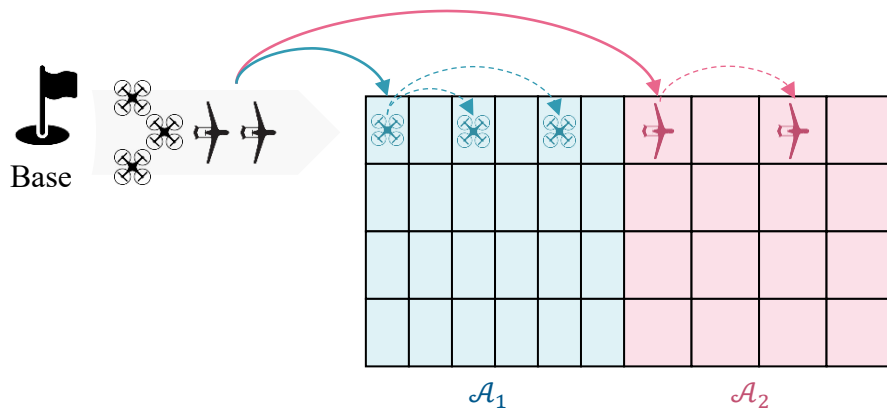


Figure 4.13: Heterogeneous canonical example

When considering two types of agents represented by the same cost equation ($c_0 = 3$, $c_v = 1$, and $c_{v_2} = 1$) and fixing a cost constraint at 100, the optimization algorithm naturally chooses to segregate the group using heterogeneity by proposing a first group of 4 agents at 2.05 m/s and a second group with 5 agents at 2.30 m/s. This heterogeneous group is able to complete the mapping mission in a time of 635 seconds at a cost of 99.93 units. On the other hand, when forcing the design optimization to use a single type of these agents, the result obtained is a group of 7 units with a velocity of 2.68 m/s for a mapping time of 672 seconds and a cost of 99.97 units. In this case, the global optimization algorithm naturally selects heterogeneity as an optimal design compared to a homogeneous one. The performance obtained by the heterogeneous group is better by 5%.

When the cost of technology is increased, the optimization tends to favor numerality (macroscopic level) to individual performance (microscopic level) and has to carefully choose the number of agents from each type: hence favoring heterogeneity. On the other hand, when technology is cheap with respect to the cost of a single agent, the optimization algorithm always has the possibility to compensate a lack of performance with the continuous microscopic variables before reaching the cost constraint. Hence, the benefits of heterogeneity in a group can be mitigated when technology is cheap to acquire with respect to the cost of individual robots.

4.1.5 Conclusions

This set of experiments focused on a canonical mapping swarming mission to prove the benefits of simultaneous or global optimization when applied to multi-robot systems. In particular, optimization schemes used so far by the research community have been compared to a simultaneous optimization algorithm. These schemes are called sequential

as they optimize the individual agents and the group operations separately, mostly due to a lack of link between microscopic and macroscopic level for advanced simulation tasks. It was shown that even though the sequential optimizers are able to get quite close to utilizing the full assigned budget, the systems they are proposing for the same cost are worse in performance than the one derived by the global optimizer. On this particular example, the performance of the swarm could be improved from 1 to 29% (16% on average) by using a simultaneous optimizer rather than sequential schemes, and for a similar cost of the system.

In fact, each optimizer allocates resources differently, focusing more or less on the microscopic level or the macroscopic level, to utilize the assigned budget as efficiently as possible. As expected, the micro-macro optimizer clearly puts emphasis on the individual performance of the agents if it is subject to the same budget constraints as the group. Hence, to help the micro-macro optimizer balance its priorities, two strategies were studied. The first one consists in making the micro step optimize a single vehicle subject to a much more realistic and limited constraint than the one used by the group. This first strategy also makes the problem get closer to real-world situations when individual vehicles have their own budget limits. The second strategy is similar to the choice made for the macro-micro optimizer: optimizing a notional swarm during the micro optimization before re-optimizing the number of agents in this swarm. In one case, the micro-macro optimizer was able to obtain the same design as the global optimizer while in the other case it obtained the same design as the macro-micro optimizer.

As stated in section 4.1.2, these results are most probably a lower bound to the possible improvements achievable in the design of multi-robot systems. Indeed, individual

robots are sequentially optimized independently of the group, but with respect to the very same mission requirements that the group is facing. In real-world situations, current suboptimal swarms have their robots optimized for different missions which are completely different than the ones the swarm will have to carry out. As a consequence, current real-world swarms are even more suboptimal than the sequential swarms used in this section to demonstrate the advantages of global optimization. Henceforth, the expected benefits from global optimization could be even greater than the numbers exposed in this section.

Moreover, global optimization truly unlocks the capabilities of heterogeneity by enabling vehicles of the same architecture to have different configurations. This is not possible when using sequential optimization since each architecture is optimized individually before being incorporated into a swarm.

4.2 Mesoscopic modeling

This section details the thought process behind the implementation of a mesoscopic model so that the steps of the methodology can be applied to different situations and mission types (such as the mapping-based canonical one defined in the introductory example). In particular, complexity is built up from the simple macroscopic model to the intricate microscopic one so that both types of models are fully understood. Building on this understanding, the compromise mesoscopic model is developed as a proper tradeoff between the other two models.

In order to conceptually answer the research question and demonstrate the idea of tradeoff between microscopic and macroscopic variables, only two design variables are varied in the models: the number of agents in the group, and their velocity. As for the

introductory example (see page 75), this presents the advantage of enabling an easy representation of the design space and the response by using tri-dimensional graphs. This is a key benefit in helping the reader and the designer comfortably visualize and understand the compromise at stake. Note that this does not mean that these variables are the only parameters of the different models. Each model requires additional design variables which are kept fixed at baseline values. In particular, these variables are set so that the performance of the models accurately represent the behavior of the same system. However, they must not be adjusted or corrected to make the models provide the same performance since this could possibly make each model represent a different vehicle. Instead, it is expected that each model will provide a different level of precision with respect to a real system and this should be accounted for.

4.2.1 Canonical mission

In order to be easily able to verify the different models against a real-world system, the chosen swarming mission must be implementable with the facilities available at the Georgia Institute of Technology. For this reason, the Robotarium platform (Figure 4.14) and its GRITBots are used for this modeling section since they are an available swarming hardware platform at the Georgia Institute of Technology. The accuracy of the different models can then be verified against the real system.

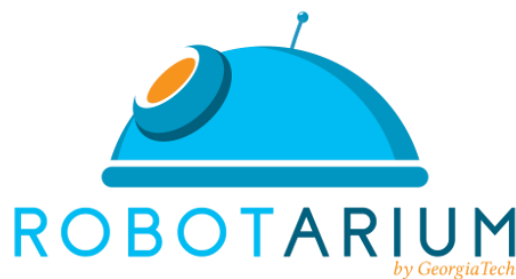


Figure 4.14: Robotarium project logo

The Robotarium project is the latest state-of-the-art multi-robot test facility with remote access capabilities. Its aim is to reduce the significant resources investments which are required nowadays to investigate multi-robot systems. By providing a remote access to its testbed, Robotarium facilitates the deployment of control algorithms on real robots instead of staying confined to simulation. Students and researchers from across the globe are able to create an account, as well as upload and test their ideas on real robotic hardware [223]. The project also includes a simulator to enable the development of the control algorithms before deploying them on the real system (Figure 4.15).

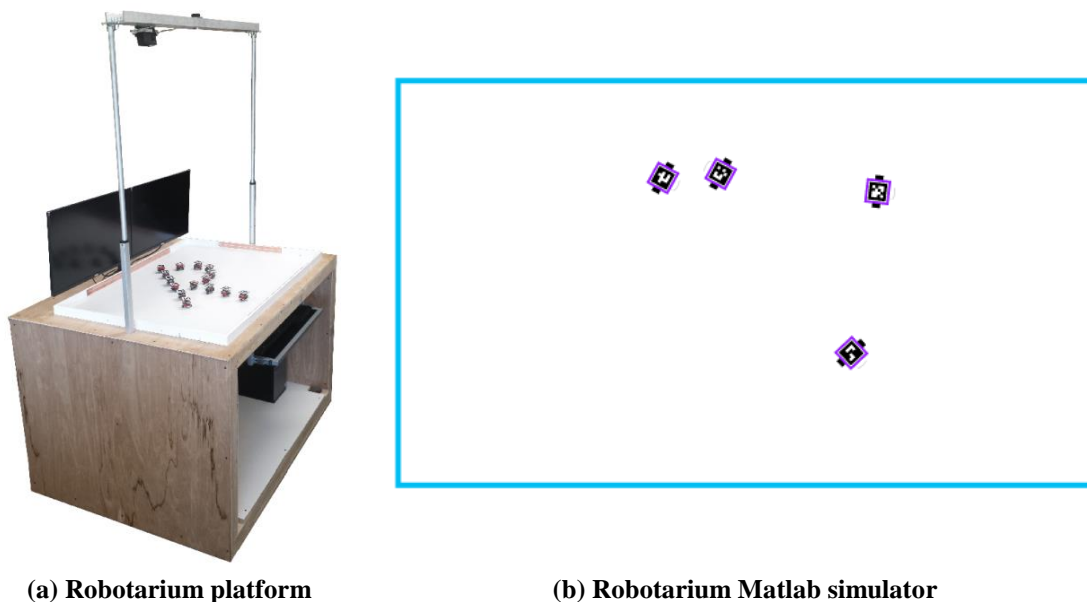


Figure 4.15: Robotarium testbed

In order to provide continuous operation, the Robotarium focuses on automated maintenance and battery charging, as well as collision-free execution of motion paths [224]. The system is comprised of the robots themselves, the position tracking system, the wireless communication hardware, the arena, and its charging system. The current testbed

dimensions are $130 \times 90 \times 180$ cm and can host up to 20 robots: the GRITSBots (Figure 4.16). These are inexpensive miniature robots equipped with a differential drive and which can be wirelessly recharged and reprogrammed. The complete design specifications are detailed in [225]. A key design element is the 400 mAh LiPo battery which provides up to 40 minutes of autonomy. The baseline linear velocity of the GRITSBots is 10 cm/s and 360 degrees/s for the rotational velocity. The corresponding maximum values are 25 cm/s and 820 degrees/s.

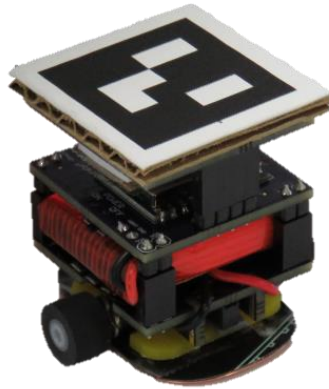


Figure 4.16: GRITBots robot platform

It is important to notice that the tracking system of the robots is centralized and omniscient so that the robots do not have to be individually equipped with sensors. This simplifies the approach and provides a layer of abstraction with respect to the localization algorithms that would have to run on the robots otherwise.

Key requirements for the mission to be implemented is that the robots should interact with each other and that the chosen design variables (velocity and number of

agents) should have an impact on the response of the system. Amongst the different example implementations available with the Robotarium simulator, the classic problem of the rendezvous is used as the canonical mission comparing the different modeling techniques. The main motivation for this choice is that the rendezvous algorithm is a problem which has been widely studied and can be easily understood and declined into different models using varying detail levels.

The rendezvous mission has the agents converge to the same final position and is usually accomplished through a network control algorithm. Formally, the 2D positions of the agents are tracked through states $\bar{X}_i \in \mathbb{R}^2, \forall i \in \llbracket 1, N \rrbracket$ with N the number of agents in the group. Each state vector contains the 2D world coordinates of a given robot. Single-integrator dynamics are used so that each robot i is controlled in terms of velocity $\dot{\bar{X}}_i = \bar{U}_i$ with $\bar{U}_i \in \mathbb{R}^2$ the control input. Using this nomenclature for the rendezvous problem, the control law has to be designed such that all robots end up at the same position (Equation 4.3).

Equation 4.3: Mathematical formulation of the rendezvous problem

$$\lim_{t \rightarrow +\infty} (\bar{X}_i - \bar{X}_j) = \bar{0}, \forall (i, j) \in \llbracket 1, N \rrbracket^2$$

The classic solution as stated in [223] is to define the control input u_i as $\bar{U}_i = \sum_{j \in N_i} (\bar{X}_j - \bar{X}_i)$ with N_i a particular set of neighbors of agent i . Usually this set of neighbors is defined as the set of agents with which agent i is able to have bidirectional

communication. Using the overhead camera tracking system, each robot has knowledge of the positions of all other robots at any point in time, hence the set N_i is the same for all robots and equal to the whole group without agent i : $N_i = \{j\}_{j \in \mathbb{N}, i \neq j}$. Using graph theory and the consensus algorithm, it can be shown that $\lim_{t \rightarrow +\infty} \bar{X}_i(t) = \frac{1}{N} \sum_{j=1}^N \bar{X}_j(0), \forall i \in \llbracket 1, N \rrbracket$. Hence, the final position of all robots is the centroid of their initial positions. An example implementation on the real Robotarium is shown on Figure 4.17.

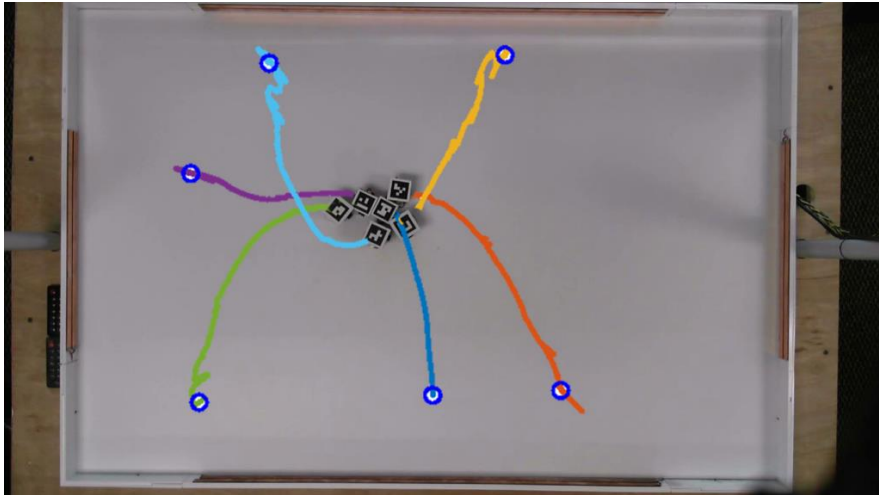


Figure 4.17: Rendezvous trajectories on Robotarium implementation [226]

The key metrics used to characterize each model versus the real system are the time taken to reach the consensus, and the final centroid position of the group. These metrics depend on both the velocity of the agents and the number of robots in the group. The consensus is considered to be reached when the average change in position for the agents is less than $1/10^{\text{th}}$ of a mm for more than 25 consecutive iterations. In order to account for complexity, an additional metric is the runtime of each model. The next subsections detail the three different models used for the rendezvous canonical mission.

4.2.2 Macroscopic model

Simplistic, fast, and easy to implement, macroscopic models are usually broken down as closed-form mathematical formulations such as differential equations. These are typically utilized to model the evolution of population movements but are not suitable for exploration-based swarming missions. A first example of macroscopic model was introduced in section 2.1 (page 75) and a more complete representation accounting for heterogeneity has been proposed in [227]. In the same fashion, a mathematical formulation is proposed here based on the theory of the rendezvous problem.

The solution provided by the consensus algorithm was detailed in the previous section and already provides closed-form formulae for the rendezvous problem. These can directly be used in the macroscopic model. In particular, the final centroid position of the group is nothing but the final position of each agent since they all have the same final position (see Equation 4.4).

Equation 4.4: First metric formula for the macroscopic model

$$\bar{X}_f = \frac{1}{N} \sum_{j=1}^N \bar{X}_j(0)$$

The other metric (i.e. time for the group to reach consensus) can then be easily computed as the time taken for the last robot to reach the final position. Assuming that the robots have the same constant velocity and that they do not have to rotate to aim at the centroid, this robot will be the furthest from the final position. This yields the closed-form formula for the second metric (see Equation 4.5).

Equation 4.5: Second metric formula for the macroscopic model

$$t_f = \max_i \left(\frac{\|\bar{X}_i(0) - \bar{X}_f\|_2}{v} \right)$$

The main assumptions used in the macroscopic model are the following:

- All robots have constant velocity v
- All dynamics neglected
- Each robot is a point mass
- No collisions
- Does not account for the orientation of the robots
- Perfect communications (the system and all robots know the positions of other robots at all times)

4.2.3 Microscopic model

Aimed at providing more accuracy, microscopic models can be as intricate as possible and include many different analyses. For instance, a complete microscopic model of an aircraft would encompass the design elements of the aircraft frame itself but also of its many subsystems, each one of these being a complex system of its own. Nevertheless, this fidelity comes at the price of computational time, development time, complexity, the difficulty to add later changes, and the need to be supported by data [228].

For the microscopic model of the GRITBots and the Robotarium system, a 6 Degrees of Freedom (DOF) simulation using the Gazebo simulator is proposed and detailed in this subsection. Gazebo includes several high-performance physics engines and

advanced 3D graphics capabilities so that the physics simulation of a robot can be completely handled from a Unified Robot Description Format (URDF) description file. This file describes the main hardware components of the robots along with their physical properties (chassis, wheels, propellers, etc.) as well as the different sensors (camera, accelerometers, microphones, etc.) as shown on Figure 4.18. This file format was created to establish a standard and many robotics suites use it such as the Robot Operating System (ROS), mostly used to code the intelligence of the robot. As a consequence, the implementation of the microscopic model for the GRITBots consists in creating the corresponding URDF file and the simulator will be able to completely simulate the physics of the robot.

```

robot name is: pr2
----- Successfully Parsed XML -----
root Link: base_footprint has 1 child(ren)
  child(1): base_link
    child(1): base_laser_link
    child(2): bl_caster_rotation_link
      child(1): bl_caster_l_wheel_link
      child(2): bl_caster_r_wheel_link
    child(3): br_caster_rotation_link
      child(1): br_caster_l_wheel_link
      child(2): br_caster_r_wheel_link
    child(4): fl_caster_rotation_link
      child(1): fl_caster_l_wheel_link
      child(2): fl_caster_r_wheel_link
    child(5): fr_caster_rotation_link
      child(1): fr_caster_l_wheel_link
      child(2): fr_caster_r_wheel_link
    child(6): torso_lift_link
      child(1): head_pan_link
        child(1): head_tilt_link
          child(1): head_plate_frame
            child(1): sensor_mount_link
              child(1): double_stereo_link
                child(1): narrow_stereo_link

```

(a) High level URDF description



(b) Actual PR2 robot [229]

Figure 4.18: PR2 robot model

To create the URDF description file for the GRITBots, the source files for the hardware of the robot are downloaded from [223]. These CAD files contain most of the physical properties, including mass and inertia, which are required to fully describe the physical components of the GRITBot robot (see Figure 4.19).

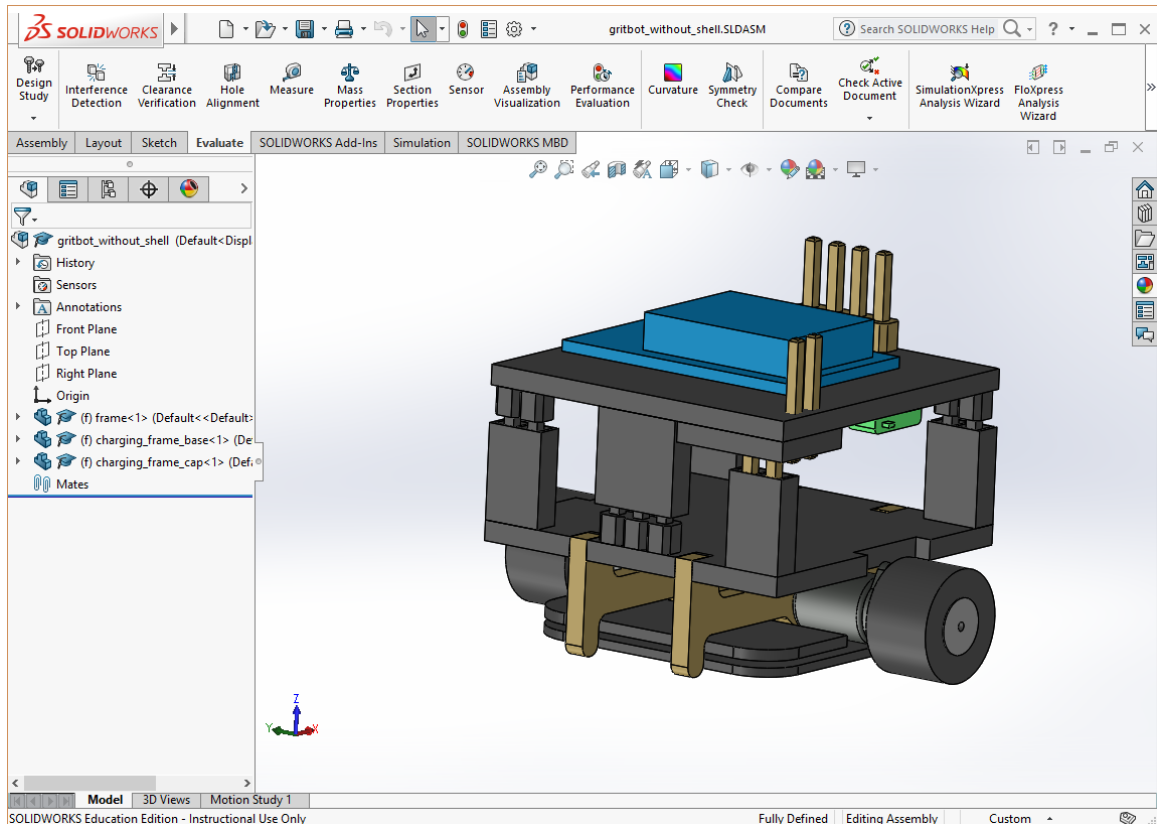


Figure 4.19: CAD model of the GRITBots

The correct dimensions, masses, and inertias are then reported in the URDF description file of the robot (see APPENDIX C page 622). The 3D model is also exported in the Collada file format to be able to be used in the Gazebo simulator (see Figure 4.20).

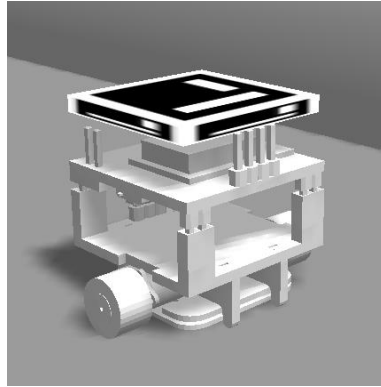


Figure 4.20: Simulated GRITBots model

This individual robot model enables to have a more precise 6 DOF simulation of the GRITBots but in order to obtain a complete microscopic model, there is a need to model the interactions of the group with the same level of detail than the one used for the individual agents. Hence, thanks to the Gazebo physics model, collisions are now modeled which adds a degree of complexity to the group interactions. In addition, the tracking system can be modeled to simulate the centralized positioning algorithm of the robots and a communication network can also be established.

The Robotarium system now being completely physically simulated, the intelligence must be implemented into ROS. This is done by instantiating several nodes where each represent one part of the behavior of the system:

- The overhead tracking system is implemented in the *tracker* node which reads images from the overhead camera simulated in Gazebo and runs the Aruco tag detection algorithm on them to estimate the poses of each robot in the arena (see Figure 4.22). The extrinsic calibration of the camera with respect to the arena enables to transform poses from the camera coordinate system to the arena

coordinate system. The Aruco tag tracking is done with the *ar_april_tag* library [230]. The tracker node then sends the poses information on the *robotarium/poses* topic for other nodes to subscribe to and use this information.

- The *consensus* node is the main node as it is the one reading the pose information given by the *tracker* node and computing the velocity commands for each of the different robots. This node implements the static consensus algorithm and publishes the control commands for each robot on their respective topics.
- The *logger* node is a simple node which registers to the poses topic and saves the pose information of the different robots in a log file, associated with the simulation time. This enables to compute the mission metrics once the simulated run is over.

The implemented nodes and their communications are summarized in Figure 4.21.

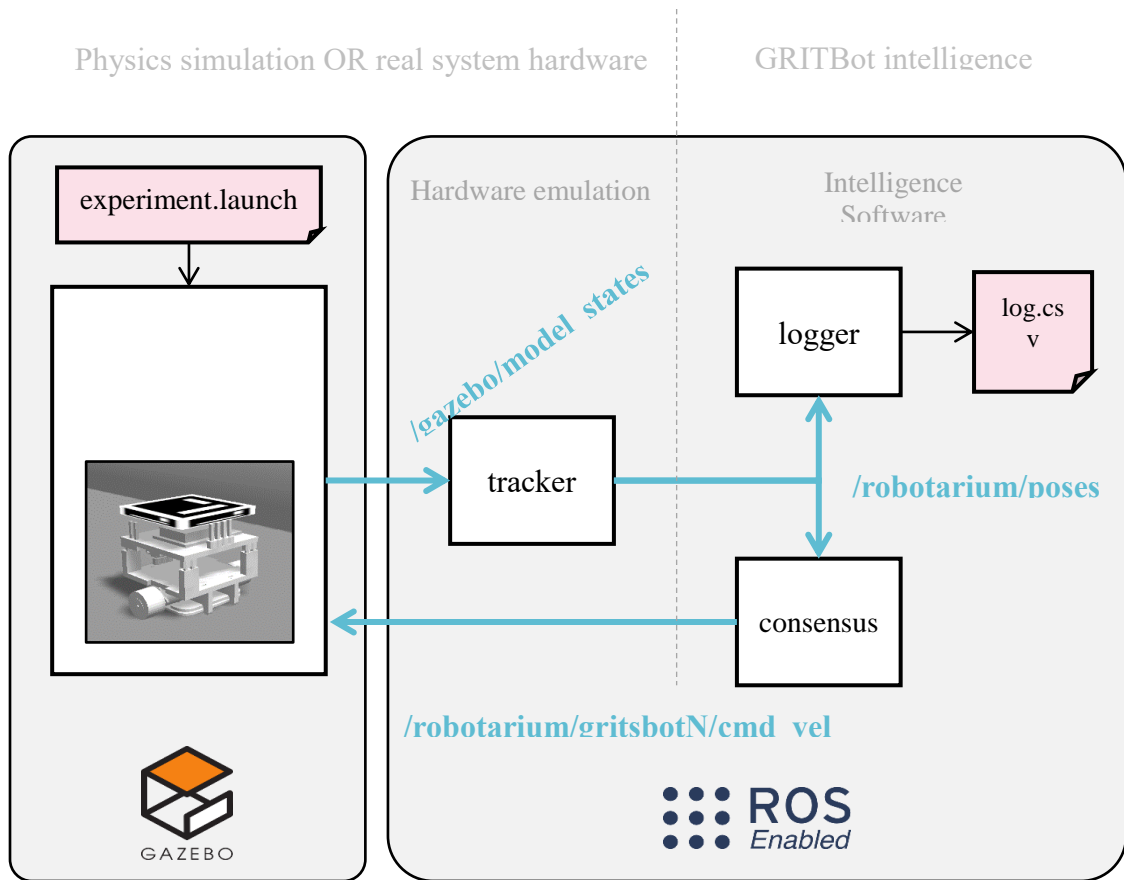
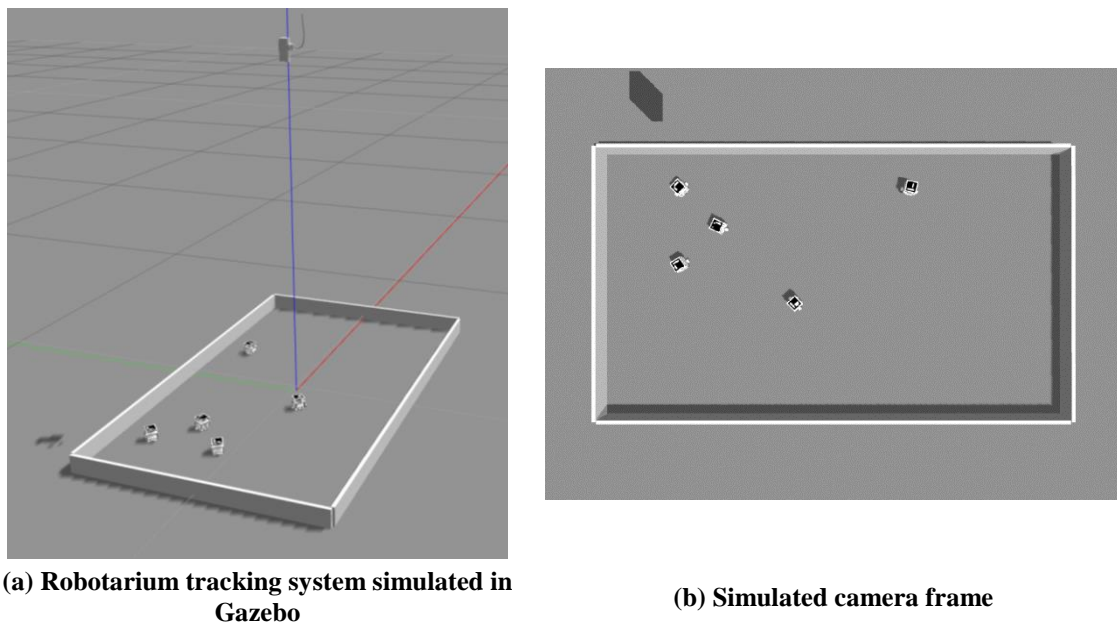


Figure 4.21: Microscopic model architecture



(a) Robotarium tracking system simulated in Gazebo

(b) Simulated camera frame

Figure 4.22: Pose tracking system simulation

To run a given experiment, the microscopic model prepares a launch file which details the simulation environment (the Robotarium arena with the tracking camera), the number of robots to spawn with their velocity, and finally the nodes to run (the *tracker*, the *consensus* node, and the *logger*).

To increase the fidelity of the microscopic model with respect to the real system, a communication topology is established between the robots. In the macroscopic model, perfect communication was assumed so that all robots are able to communicate with all other robots when computing their consensus velocity. This assumption is here reviewed to limit the number of neighbors a given agent is able to communicate with. The set N_i of neighbors of agent i is generated by an undirected graph $G(V, E)$ constructed in a way that $j \in N_i \Leftrightarrow (i, j), (j, i) \in E$ with V the vertices and E edges of graph G . An example of the Laplacian matrix of G is given by Equation 4.6.

Equation 4.6: Laplacian matrix for N=5 agents

$$\begin{bmatrix} 2 & -1 & 0 & 0 & -1 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ -1 & 0 & 0 & -1 & 2 \end{bmatrix}$$

This form of the matrix implies that any agent is able to communicate with 2 other robots only.

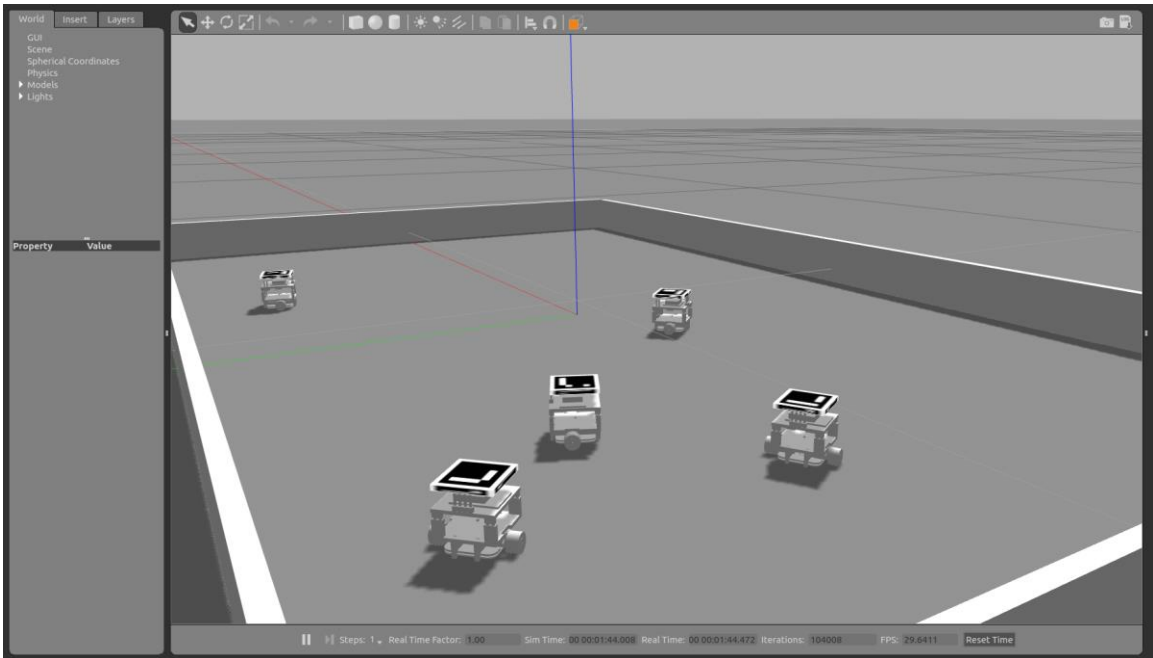


Figure 4.23: Robotarium arena and GRITBots in the Gazebo simulator

The ROS-Gazebo implementation constitutes the microscopic model of the Robotarium and the assumptions associated with it are the following:

- Kinematics and dynamics are simulated based on 6 degrees of motion including mass and inertia models. Robots evolve in a 3D environment (see Figure 4.23).
- Motion is ensured through a planar differential drive plugin accounting for friction.
- The interactions between the agents are accurately detailed with:
 - Collisions modeling through cubic collision meshes elements. In addition, the control algorithm still includes safety radiuses and barrier certificates.
 - The poses of the agents are now acquired through a simulated overhead camera-based system tracking Aruco tags. This makes the centralized pose acquisition imperfect and prone to noise as in the real system.

- The communication range is now limited and an agent can only communicate with neighbors induced by a given communication topology. In this case, the topology is static and represented by an undirected graph and its Laplacian matrix. The graph is designed so that each agent is able to communicate only with two other robots.

4.2.4 Mesoscopic model

Now that macroscopic and microscopic models have been developed and fully understood, a compromise mesoscopic model has to be constructed. The resulting model is expected to have a decent precision and remain computationally inexpensive to run. For instance, by looking at the different fidelity of the aircraft models given in [228], they can be categorized as shown on Figure 4.24.

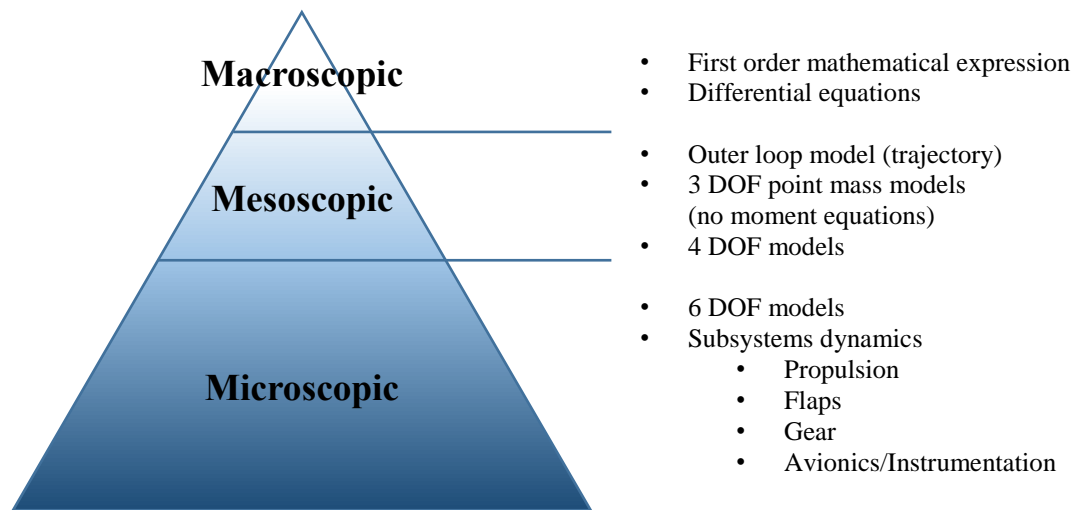


Figure 4.24: Different levels of detail for aircraft models

As the level of detail decreases when going from microscopic to macroscopic modeling, design variables are lost, giving rise to a number of simplifications and

assumptions in the original models. As mentioned in section 2.2.2.2, this generalization is often achieved by reducing the level of detail on the group modeling while maintaining a sufficient level of detail for the models of individual agents. Given that the level of detail of microscopic models can always be adjusted, and that mesoscopic models can generally be considered as surrogates of microscopic models, it is important to clearly define the bounds considered. Indeed, for a given level of detail, it is always possible to go one level lower. Hence, for establishing these models, the lowest level of detail achievable is considered to be the theoretical and analytical derivation of the static consensus, and the highest level of detail is considered to be the real system implemented on the Robotarium.

In the case of the GRITSBots and the Robotarium, the group dynamics are acquired through the camera tracking system which provides the position of all robots at a given point in time. This part of the microscopic model can be simplified by simply assuming that all positions are known instead of constantly computing them. This step removes quite a complex layer of the microscopic model by eliminating all the computer vision components. The delay of 33 ms required to compute the poses of the robots is hence removed. Indeed, [225] showed that the tracking system is limited to a framerate of 10 fps for 5 robots and 6.5 fps for 25 robots.

Unicycle model: a second step in obtaining a mesoscopic model for the GRITBots and the Robotarium can be to use a simplified physics model for the dynamics of the robot. Indeed, while the microscopic model uses a complete 6 DOF model, inspiration can be taken from Figure 4.24 to implement a simpler 3 DOF representation for the mesoscopic level of detail.

This has the benefit of still representing the dynamics of the robot (as opposed to the macroscopic model) while reducing the complexity of the model. The 3D simulation of the microscopic model now becomes a 2D simulation using the differential drive dynamics, the model used in the Robotarium simulator. Also known as unicycle dynamics, they represent a robot moving in a two-dimensional world with a given forward velocity but no lateral motion. The controls of such a robot are the norm of the forward velocity vector and its azimuth in the world (see Figure 4.25).

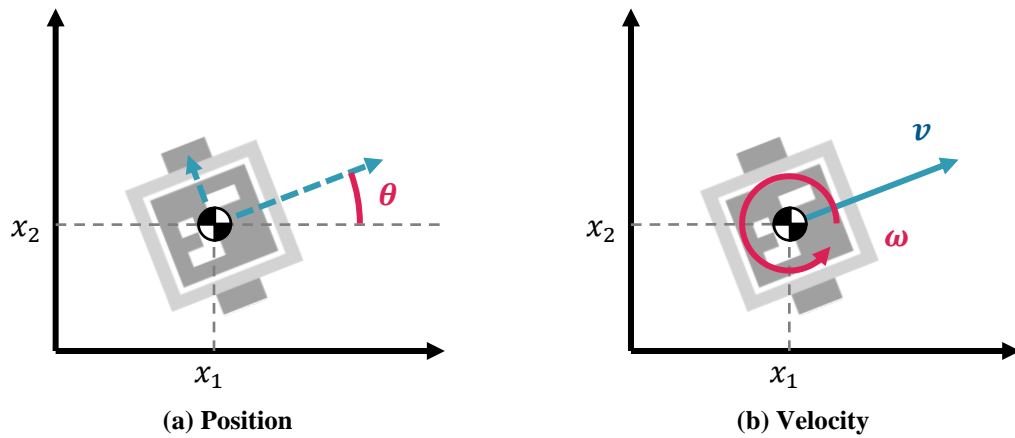


Figure 4.25: Unicycle model representation

The kinematic model of a differential drive robot is given by Equation 4.7.

Equation 4.7: Kinematic model of a unicycle robot

$$\bar{\dot{x}} = \begin{Bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{\theta} \end{Bmatrix} = \begin{Bmatrix} v \cos \theta \\ v \sin \theta \\ \omega \end{Bmatrix} = \begin{bmatrix} \cos \theta & 0 \\ \sin \theta & 0 \\ 0 & 1 \end{bmatrix} \begin{Bmatrix} v \\ \omega \end{Bmatrix}$$

Where $[x_1, x_2, \theta]$ is the position and orientation of the robot in the world reference frame and $[v, \omega]$ the linear and angular velocities.

Controls: the control algorithm used for the GRITBots and given in [225] uses feedback linearization. The idea is to control a point $[x'_1, x'_2]$ situated in front of the robot and offset by a fixed length λ which corresponds to an equivalent gain for the controller. The control is then done by adjusting the velocity of this point $[v_{x_1}, v_{x_2}]$ in the world coordinates. This can be given by the rendezvous command inputs for instance. Hence, these single-integrator dynamics $[v_{x_1}, v_{x_2}]$ have to be translated to the unicycle dynamics $[v, \omega]$ which can be directly interpreted by the robot model (see Equation 4.7). The dynamics of the controller are computed as follow:

Equation 4.8: Dynamics of the Gritbots controller

$$\begin{aligned}
\begin{Bmatrix} x'_1 \\ x'_2 \end{Bmatrix} &= \begin{Bmatrix} x_1 \\ x_2 \end{Bmatrix} + \lambda \begin{Bmatrix} \cos \theta \\ \sin \theta \end{Bmatrix} \\
\Rightarrow \overline{X}' &= \overline{X} + \lambda \begin{Bmatrix} \cos \theta \\ \sin \theta \end{Bmatrix} \\
\Rightarrow \overline{\dot{X}}' &= \overline{\dot{X}} + \lambda \frac{d}{dt} \begin{Bmatrix} \cos \theta \\ \sin \theta \end{Bmatrix} \\
\Rightarrow \overline{\dot{X}}' &= \overline{\dot{X}} + \lambda \dot{\theta} \begin{Bmatrix} -\sin \theta \\ \cos \theta \end{Bmatrix} \\
\Rightarrow \overline{\dot{X}}' &= \begin{Bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{Bmatrix} + \lambda \dot{\theta} \begin{Bmatrix} -\sin \theta \\ \cos \theta \end{Bmatrix} \\
\Rightarrow \overline{\dot{X}}' &= v \begin{Bmatrix} \cos \theta \\ \sin \theta \end{Bmatrix} + \lambda \omega \begin{Bmatrix} -\sin \theta \\ \cos \theta \end{Bmatrix} \\
\Rightarrow \overline{\dot{X}}' &= \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{Bmatrix} v \\ \lambda \omega \end{Bmatrix} \\
\Rightarrow \overline{\dot{X}}' &= \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & \lambda \end{bmatrix} \begin{Bmatrix} v \\ \omega \end{Bmatrix} \\
\Rightarrow \begin{Bmatrix} v_{x_1} \\ v_{x_2} \end{Bmatrix} &= R(\theta)S(\lambda) \begin{Bmatrix} v \\ \omega \end{Bmatrix} \\
\Rightarrow \begin{Bmatrix} v_{x_1} \\ v_{x_2} \end{Bmatrix} &= G(\theta, \lambda) \begin{Bmatrix} v \\ \omega \end{Bmatrix}
\end{aligned}$$

The diffeomorphism G gives the relation between the single-integrator and the unicycle dynamics. By computing its inverse, the control input of the robot can be expressed as a function of the rendezvous single-integrator commands:

Equation 4.9: Inverse control commands for the Gritbots

$$\begin{aligned} \begin{Bmatrix} v \\ \omega \end{Bmatrix} &= G^{-1}(\theta, \lambda) \begin{Bmatrix} v_{x_1} \\ v_{x_2} \end{Bmatrix} \\ \Rightarrow \begin{Bmatrix} v \\ \omega \end{Bmatrix} &= S^{-1}(\lambda)R^{-1}(\theta) \begin{Bmatrix} v_{x_1} \\ v_{x_2} \end{Bmatrix} \\ \Rightarrow \begin{Bmatrix} v \\ \omega \end{Bmatrix} &= \begin{bmatrix} 1 & 0 \\ 0 & 1/\lambda \end{bmatrix} \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix} \begin{Bmatrix} v_{x_1} \\ v_{x_2} \end{Bmatrix} \end{aligned}$$

This command vector can then directly be used in Equation 4.7. The same control technique is used in the microscopic model.

System identification: now that the dynamics and controls of the robot have been established, the gap between mesoscopic simulation and real-system must be bridged to obtain a validated model [224]. Velocity measurements \hat{x}_i of the model velocities \dot{x}_i are made on the real system $\forall i \in \{1,2,3\}$. A linear relation is assumed between the model and the observations, as shown in Equation 4.10.

Equation 4.10: System identification model

$$\bar{\hat{x}} = \begin{bmatrix} \alpha_1 & 0 & 0 \\ 0 & \alpha_2 & 0 \\ 0 & 0 & \alpha_3 \end{bmatrix} \bar{\dot{x}}$$

Where the α 's are linear regression coefficients. With a measurement campaign of 30,000 measurements carried out in [224], measurement vectors $\overline{\widehat{X}}_i \in \mathbb{R}^{30,000}, \forall i \in \{1,2,3\}$ are obtained and used in a least squares linear regression scheme to compute the regression coefficients: $\alpha_i = \left(\overline{\widehat{X}}_i^T \overline{\widehat{X}}_i \right)^{-1} \left(\overline{\widehat{X}}_i^T \overline{\widehat{X}}_i \right), \forall i \in \{1,2,3\}$. The results are summarized in Equation 4.11 with $\overline{\widehat{x}}$ given by Equation 4.7.

Equation 4.11: Final values for system identification

$$\overline{\widehat{x}} = \begin{bmatrix} 0.8645 & 0 & 0 \\ 0 & 0.8119 & 0 \\ 0 & 0 & 0.4640 \end{bmatrix} \overline{\widehat{x}}$$

Collisions modeling: collisions are indirectly modeled through barrier certificates which avoid collisions between the robots by modifying the user control inputs if some robots are too close to each other. The barrier certificate method prevents collisions and is minimally invasive since it minimizes the deviation between the newly computed safe control command and the original command provided by the rendezvous formulae. This minimization is subject to safety constraints which enforce a minimum distance between the robots. Additional constraints ensure that the output velocity is bounded within the physical limits of the robots. The minimization is performed with quadratic programming and a full derivation of the barrier-certificate-compliant controls is given in Appendix A.3 (see page 445). [224] showed that this collision avoidance scheme is able to run in real-time on a decentralized system with up to 100 agents at 185 Hz.

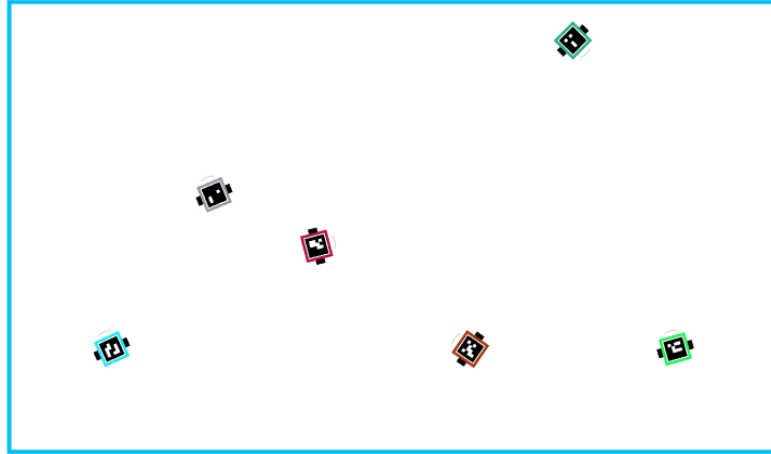


Figure 4.26: Graphic interface of the mesoscopic model

This mesoscopic model corresponds to the implementation of the Robotarium simulator and the different assumptions used in its elaboration are listed here below:

- Kinematics are simulated based on 3 degrees of motion (position and orientation) with a unicycle motion model. Robots evolve in a 2D environment (see Figure 4.26).
- Dynamics (mass, inertia) are neglected.
- Friction is neglected.
- Collisions are indirectly modeled through safety radiuses and barrier certificates.
- Perfect communications are assumed so that the system and all robots know the positions of other robots at all times. The tracking and pose estimation system is abstract. Robots poses are known exactly and for each time step.

4.2.5 Verification and validation

The verification and validation of simulation models generally consists in comparing the performance of the models with respect to the real system. Given that this

section focuses on modeling, the verification and validation portion is joint with the characterization part and presented in the next section. This latter presents the different experiments required to fully characterize each model.

4.2.6 Characterization

Based on the example canonical mission, this section has demonstrated how to progressively construct a mesoscopic model by leveraging macroscopic and microscopic models (Figure 4.27).

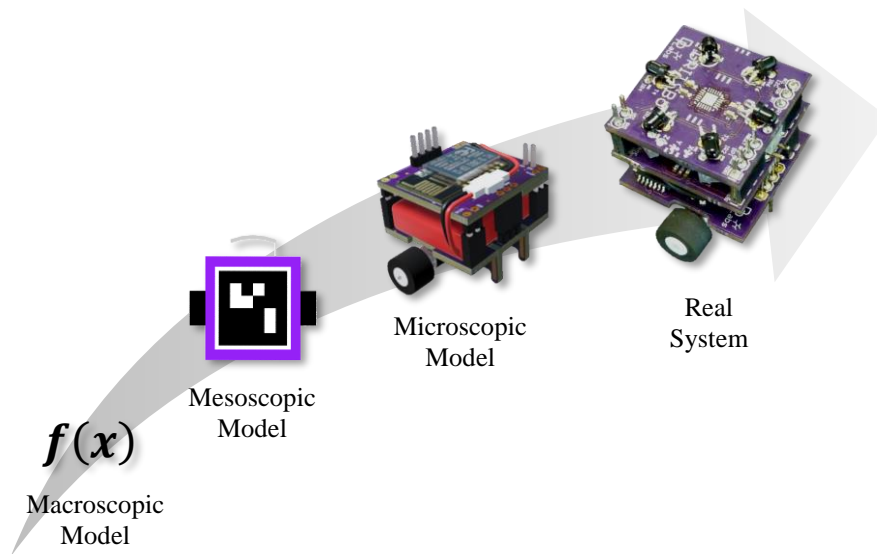


Figure 4.27: The increasing level of detail of the implemented GRITBot models

The three models are now compared with each other in terms of the different evaluation criteria established for experiment 2. Due to the hardware limitations inherent to the current Robotarium platform, the velocity of the robots is set to evolve from 1 to 10 cm/s by increments of 1 cm/s and the number of robots is changed from 2 to 5. This design of experiments hence generates 40 evaluation points for each one of the models.

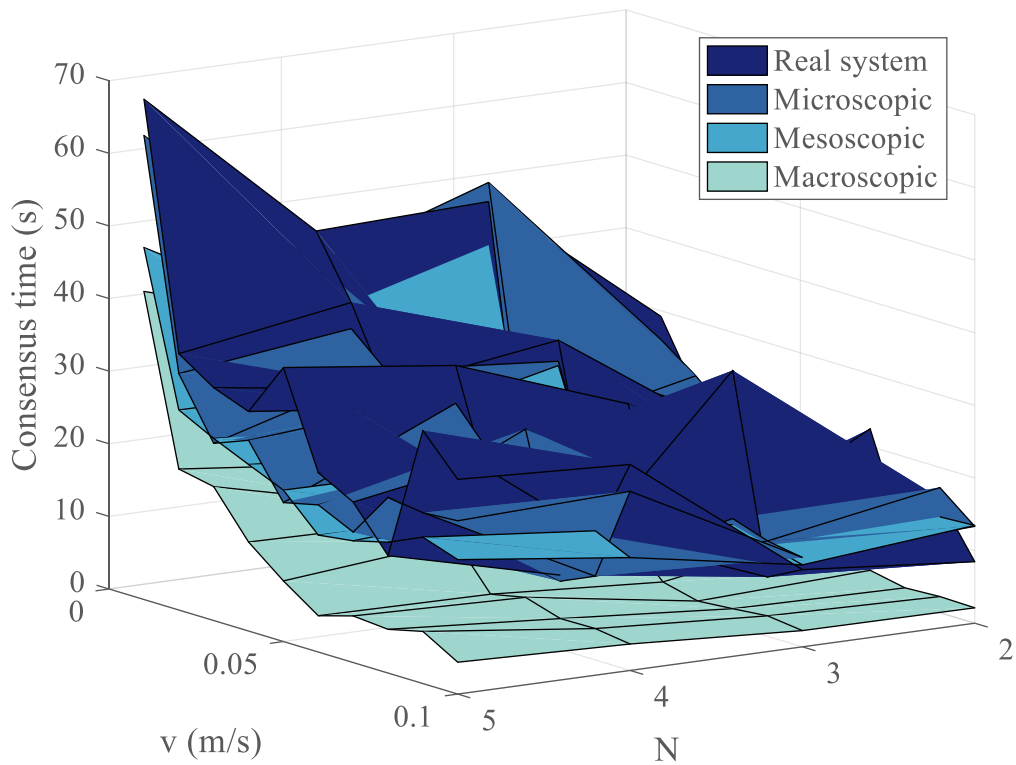


Figure 4.28: Consensus time metric

By first looking at the consensus time response (Figure 4.28), an initial comparison of the models can be drawn. Observing the overall evolution of the response, it appears that the consensus time decreases as the velocity is increased and the number of robots decreased. However, note that the initial positions of the robots are completely aleatory for each data point, hence no conclusions can be drawn from the general shape of the response since all experiments are unrelated to each other.

The three models seem to superimpose on each other, each time getting closer to the real system response as the level of detail is increased from macroscopic to microscopic. In particular, all models seem to be overconfident in the performance of the system by underestimating the time it would take to achieve consensus: all response

surfaces lie underneath the response surface of the real system. As could have been expected, the macroscopic model is being the most optimistic by predicting quite low consensus times when compared to the real system. This is mostly due to the fact that all dynamics and interactions between the agents are neglected and these are the most prominent factors in the performance losses of the system. Indeed, the robots will first have to turn and align with their trajectory to reach consensus, then they will possibly have to interact with other robots in order to avoid collisions on the way to consensus. These two factors contribute to lengthening the time required for the swarm to reach static consensus. Then, as these factors are more and more accounted for in the mesoscopic and microscopic models, the responses get closer to the performance of the real system. This is consistent with what was expected in the validation criteria of experiment 2.

As the level of detail is increased, the models are not as optimistic as the macroscopic models and the different responses start to intersect with sometimes pessimistic predictions. This can be seen more clearly on Figure 4.29.

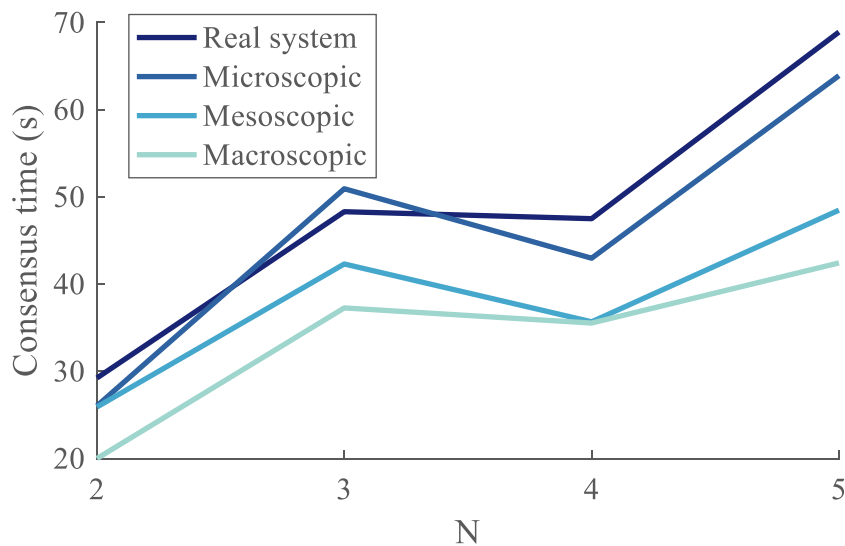


Figure 4.29: Section view of consensus time response for $v = 1\text{cm/s}$

Focusing now on the spatial precision of the different models, a previous validation of the Robotarium simulator showed that it is on average precise within 5 mm of actual robot trajectories [224], hence validating the mesoscopic model. However, the performance of the two other models is left to be evaluated and validated. An example of the different trajectories is presented on Figure 4.30.

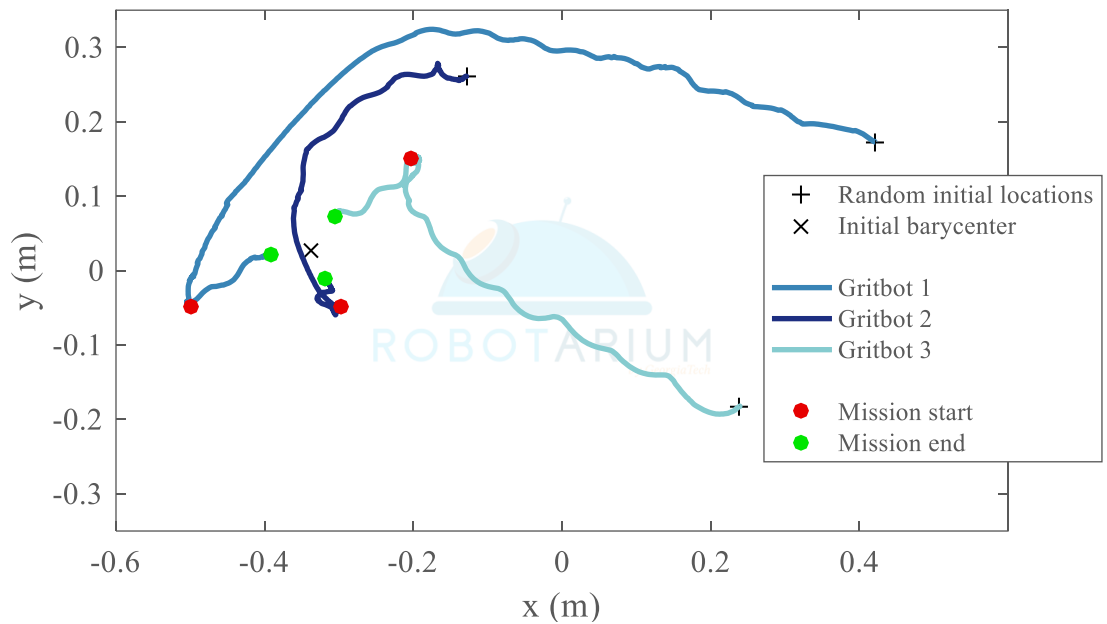


Figure 4.30: Robot trajectories for $N=3$ and $v=8$ cm/s

The Gritbots start either from their charging stations or from random initial locations based on where the operator places them (red circles). The parking controller is then activated to make the robots reach a set of initial conditions (magenta circles) which was randomly generated for the design of experiments. For a given number of robots and a given velocity, these initial conditions are the same for each model evaluation. Then, the consensus algorithm is started from these initial conditions until the Gritbots reach

consensus (green circles). The theoretical consensus location is the barycenter of the initial positions and is represented by the black cross.

A statistical analysis of the results over the complete set of 40 experiments is carried out and the summary is presented on the figures here below.

The first metric to be evaluated over the full set of 40 experiments is the location of the consensus. The macroscopic model uses the exact theoretical formula for the position of the consensus but all models are compared with respect to the performance of the real system. The results are presented on Figure 4.31.

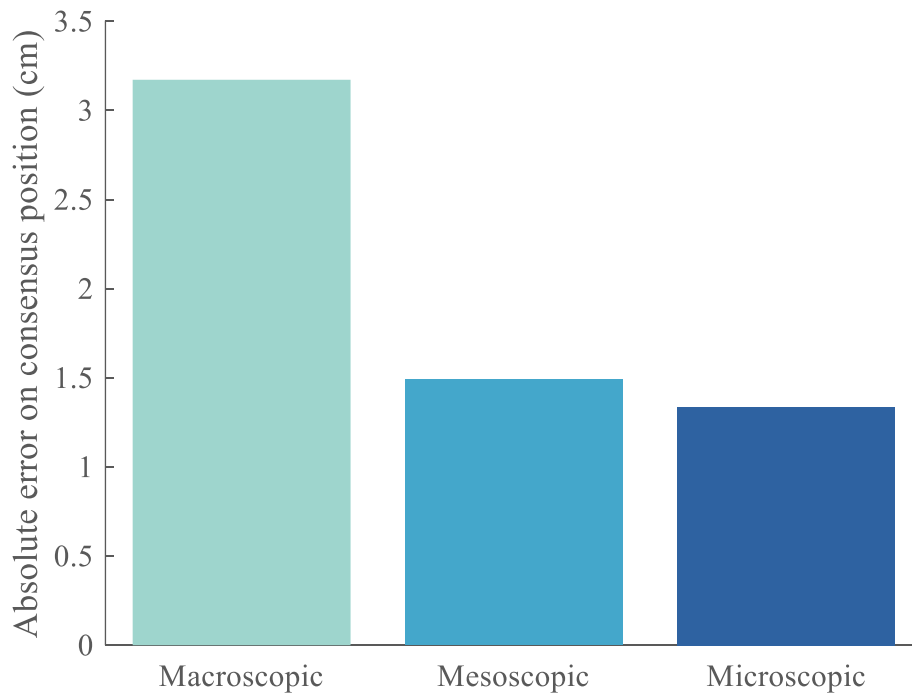


Figure 4.31: Precision of the models on the consensus position metric

Representing the error of the real system with respect to the theoretical values, the macroscopic model exhibits an error of 3.17 cm. The microscopic model is the closest to the performance of the real system with an error of 1.34 cm. Finally, and as expected by the motivations of mesoscopic modeling, the mesoscopic model lies in between with an absolute position error of 1.49 cm.

The second metric to be evaluated is the time required by the swarm to reach a consensus (see Figure 4.32).

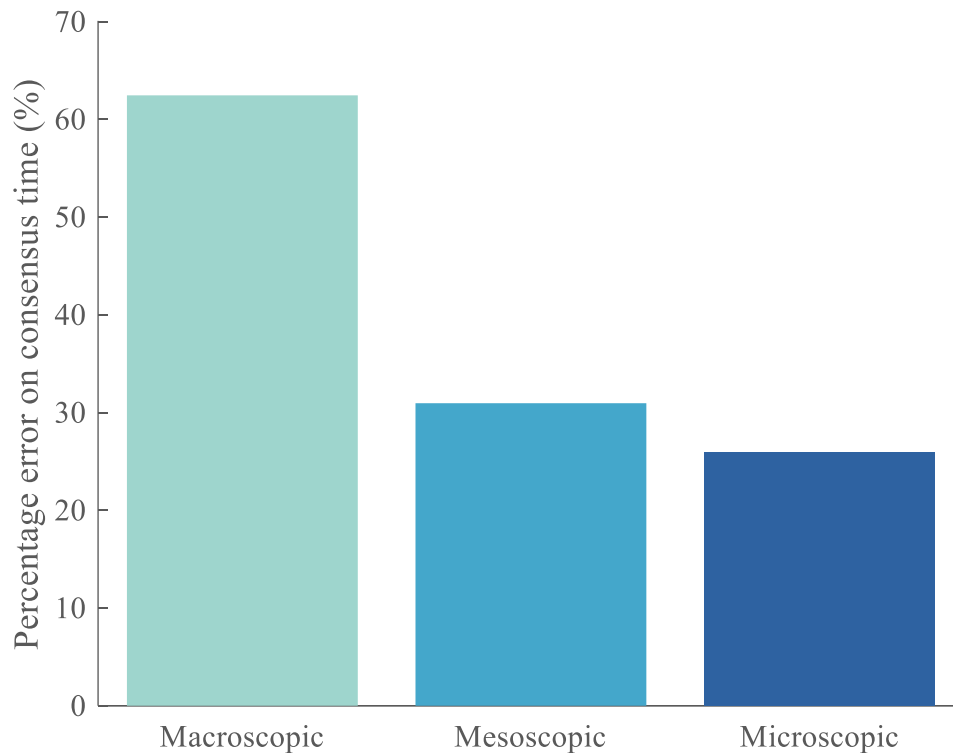


Figure 4.32: Precision of the models for the consensus time metric

The trend remains very similar to what was observed on Figure 4.32 with the macroscopic model exhibiting the most error with 62.5%, then the mesoscopic model with 31.0%, and finally the microscopic model with 26.0% of error on the consensus time. Once again, these results correspond to the behavior which was expected from the mesoscopic model: a performance between those of the macroscopic and the microscopic models. Indeed, by neglecting all dynamics, the macroscopic model exhibits a high error on the time metric which is directly linked to dynamics. Thanks to a more complete model and without being as intricate and complex as the microscopic model, the mesoscopic one is able to bring this error down to 31.0% with a simple set of assumptions.

Finally, the last metric considered for the choice of the model is the run time required by each model. It has been established in the previous paragraphs that the mesoscopic model seems to provide good accuracy with simple assumptions but it is essential to know whether this type of modeling is computationally efficient enough to justify its use for conceptual design phases. The average runtime of the different models is represented on Figure 4.33 and also compared with the time required for an experiment on the real system to be completed.

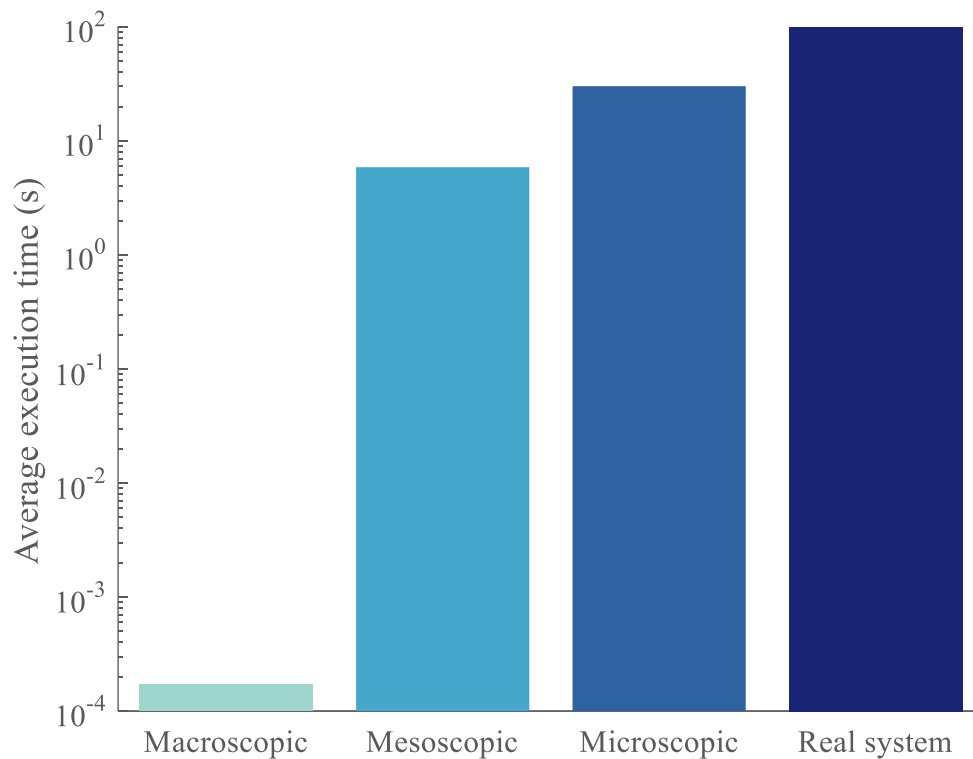


Figure 4.33: Average runtime of the different models

Once again, the numbers of the mesoscopic model seem to lie between those of the macroscopic and the microscopic model, validating our criteria for the experiment. Relying only on a couple mathematical formulae which are very quick to evaluate, the macroscopic model runs on average in 0.17 milliseconds. With dynamics involved and actual simulation of the path of the robots, the mesoscopic model runs on average in 5.88 seconds. Regarding the microscopic model, complete collision modeling and pose tracking are now included, which require more computational resources. This model runs on average in 30.16 seconds. Finally, the real system takes 99.78 seconds to run one experiment on average.

These runtimes do include the overhead setup time necessary to launch the simulators and accomplish the required preparation tasks before the consensus mission.

This includes parsing the configuration files, spawning the robots with appropriate configurations, and setting up the environment. Note that for the real system, there is an additional overhead time necessary for the parking controller in order to make the robots reach their initial positions within a certain tolerance margin.

Going back to the validation criteria established in section 2.2 for experiment 2, it is possible to conclude on the suitability of mesoscopic modeling for the conceptual design of multi-robot systems. First, it was shown that mesoscopic modeling can indeed be applied to multi-robot problems and no failure point was observed since the mesoscopic model always appeared more precise than the macroscopic model but also always faster than the microscopic one. In more detail, it was found on the studied particular case that the mesoscopic model ran on average five times faster than the microscopic model while not having more than 20% error with it. In particular, the mesoscopic model showed 31.0% of error with respect to the real system on the consensus time metric while the much more developed microscopic model showed 26.0% of error. This order of magnitude is typically what is observed in the conceptual design phases, hence confirming the appropriateness of mesoscopic modeling for the scope of the present research. Surprisingly, the performance of the mesoscopic model in terms of precision is not exactly in between those of the macroscopic and the microscopic one (see Figure 4.34).



Figure 4.34: Notional representation of the precision of mesoscopic modeling

Indeed, the mesoscopic model is found to have an accuracy much closer to the microscopic model than the macroscopic one, while being much less computationally expensive. These observations validate hypothesis 2 and position mesoscopic modeling as the ideal candidate for the rapid and precise exploration of gigantic multi-architecture and multi-level design spaces.

CHAPTER 5

MULTI-ARCHITECTURE MULTI-LEVEL DESIGN SPACE EXPLORATION

Second main axis of this research, the exploration of multi-architecture and multi-level design spaces, such as the ones encountered with swarming systems, remains a challenge mostly due to the gigantic size of the design space. Such design spaces are multi-architecture since several architectures of robots are considered to be integrated in the swarm, and multi-level since design choices have to be performed at the macroscopic level (the group), and propagated to the microscopic level (the agents). It was notably established in section 1.4.2 and the second chapter that the design space of an individual robot is somehow multiplied by the possibilities available at the macroscopic level, tremendously expanding the overall design space. This section hence aims at elaborating a pertinent design space exploration technique to facilitate the optimization of multi-robot systems by focusing on the third research question:

Research question 3

How can current conceptual design methods be adapted to account
for multi-architecture multi-level design space exploration?

This research question was later decomposed into two complementing perspectives: the generation of alternatives and the optimization of configurations. The following sections detail each of these in their implementation and their results.

5.1 Generation of alternatives: the tree of reduced morphological matrices

Before optimizing swarm configurations, they have to be properly generated so that a large portion of the design space is actually covered by the exploration technique. This concern originated a first sub research question:

Research question 3.1

How can we systematically generate all feasible alternatives in a multi-architecture and multi-level design space for further optimization?

This subsection answers this question by studying in detail the corresponding hypothesis:

Hypothesis 3.1

IF a tree of reduced morphological matrices is used
THEN all feasible alternatives can be generated in a multi-architecture and multi-level design space for further comparison and optimization

It starts by recalling the theory behind the proposed morphological tree (see section 2.3.1.2 page 141). This subsection then constructs notional morphological matrices of different architectures before implementing and applying the proposed approach of the morphological matrix tree summarized on Figure 2.23 page 144.

5.1.1 Step 1: morphological reduction

With a constant need of reducing the complexity of design optimization, it is common to separate optimizers and let them handle different disciplines or levels of the optimization for instance. The problem with this approach is that each one of the optimizers needs to handle different design variables. Hence the main idea introduced by [173] is to accelerate the design optimization process by tackling it from the morphological approach side, hence reducing the number of available discrete options to be optimized: there is a need to regroup design variables at the morphological level. For instance, instead of considering different optimizers which would handle different types of wings, it is possible to regroup straight wings, delta wings, and swept wings into a same optimizer by considering design variables such as surface area, aspect ratio, sweep angle, and other additional variables. In the same fashion, architectures such as tricopters, quadcopters, hexacopters, and other octocopters, could be regrouped in a modular “multicopter” architecture and optimizer which would instead consider the number of rotors as a design variable. Discrete design choices are hence removed from the discrete morphological analysis and directly incorporated into the optimizers. This helps reducing the number of optimizers required for the given architectures. Based on the approach proposed by [173] to reduce the number of optimizers, morphological reduction is composed of four steps:

- 1) **Generate alternatives:** this first step uses the morphological approach described earlier where functions or features are listed in the rows of a morphological matrix and corresponding possible options are enumerated in the columns. All alternatives

are then generated by using a full factorial approach taking every possible option for every feature (see Equation 5.1).

Equation 5.1: Number of alternatives from morphological analysis

$$N_{alt} = \prod_{i \in rows} N_{options}(i)$$

Where $N_{options}(i)$ is the number of options available for a given row i .

- 2) **Ensure feasibility:** an alternative can be generated only if all of its options for each feature are compatible with each other. In order to check compatibility between options, a square matrix called compatibility matrix is used. Such a matrix indicates for a given option whether it is compatible or not with all other possible options. An example of such a matrix is provided in Figure 5.1.

	O_1	O_2	O_3	O_4	O_5	O_6	O_7	O_8
O_1	1	0	1	0	0	1	0	0
O_2		1	0	1	0	0	0	1
O_3			1	1	1	1	0	0
O_4				1	0	1	0	0
O_5					1	1	1	1
O_6						1	0	1
O_7							1	0
O_8								1

Figure 5.1: Example of compatibility matrix

It can be seen that option O_1 is compatible with itself and O_3 for instance, but not with O_4 . Each option being compatible with itself, the diagonal is always set to ones. Since the matrix is symmetric, only one half of it needs to be documented. Moreover, the morphological matrix is constructed so that only one option can be chosen per feature. Table 5.1 shows an example of how this can be achieved when several options could be concurrently chosen for a given feature.

Table 5.1: One option chosen per feature

Features		Options	
Sensors			
Imaging	Mono	RGB camera	Mono
			+ RGB camera

In the previous table, the imaging sensor options are a mono channel camera and a RGB camera. However, it could be possible to consider putting both on a vehicle for imaging purposes: the mono channel sensor could be used for rapid optical flow processing while the color camera could be used for more advanced image processing. Hence, to satisfy the assumption that for each feature only one option is chosen, the alternative of having both the mono and RGB sensor is created as an additional option. This simplification can be automatically implemented in the methodology as long as rows which have possible combinatorial choices of options are identified.

If this assumption is used, the approach is simplified and the total number of entries of the compatibility matrix which must be filled-in by the designer is reduced (see Equation 5.2).

Equation 5.2: Number of filled elements in compatibility matrix

$$N_{compatibility} = \frac{N_{options}(N_{options} - 1)}{2} - \left(\sum_{i \in rows} \frac{N_{options}(i)[N_{options}(i) - 1]}{2} \right)$$

This formula consists of a first term being the number of elements strictly above the diagonal (yellow), minus all blocks of options pertaining to the same feature (red blocks, see Figure 5.2). Options O_2 and O_3 describe the same feature, as well as O_4 , O_5 , and O_6 . This can possibly represent memory savings in data structures.

	O_1	O_2	O_3	O_4	O_5	O_6	O_7	O_8
O_1	1	0	1	0	0	1	0	0
O_2		1	0	1	0	0	0	1
O_3			1	1	1	1	0	0
O_4				1	0	0	0	0
O_5					1	0	1	1
O_6						1	0	1
O_7							1	0
O_8								1

Figure 5.2: Number of filled elements in compatibility matrix

- 3) **List variables:** for each feature, the design variables which are used by at least one option are stored and associated with their respective option(s).
- 4) **Regroup options:** the options that are described by the same design variables are regrouped into one option. Finally, features which are described by a unique set of variables are removed from the morphological matrix. The analyses associated with such simplifications are transferred to the optimization algorithms. This can be represented in Table 2.8 (page 140), Table 2.9 (page 140), and Figure 5.3 here below.

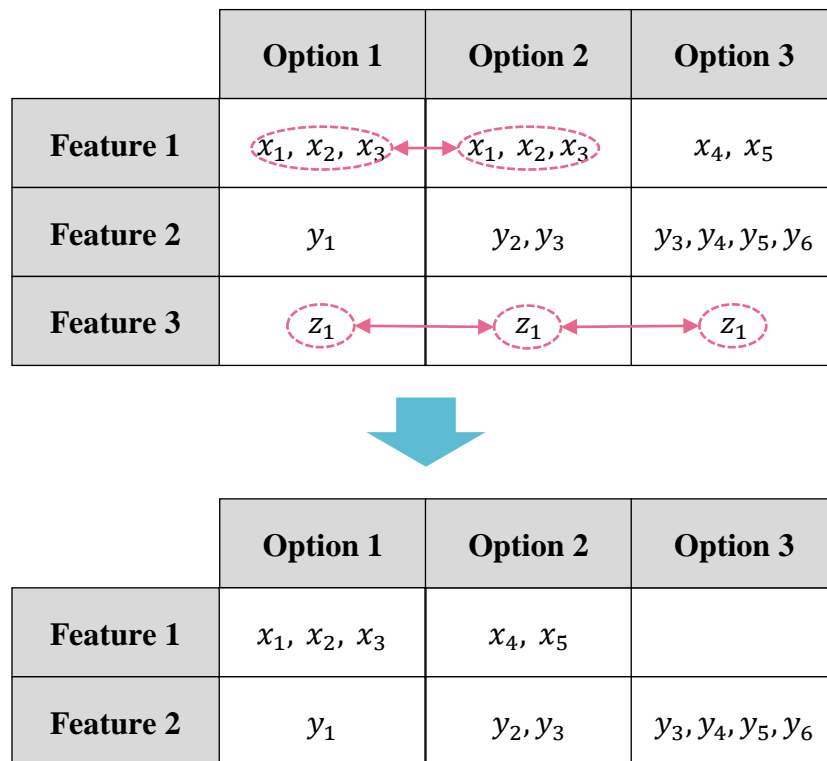


Figure 5.3: Regrouping of options

5) **Regroup architectures:** all compatible alternatives are then generated using the morphological and the compatibility matrices. Finally, amongst those alternatives, the ones that are described by unique sets of design variables are declared as architectures. In particular, alternatives having the same set of design variables will be regrouped (Table 5.2).

Table 5.2: Grouping of alternatives into architectures

Alternatives						
$\begin{Bmatrix} x_1 \\ x_2 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ z_1 \\ z_2 \end{Bmatrix}$	$\begin{Bmatrix} x_1 \\ x_2 \\ x_3 \\ y_5 \\ y_6 \\ y_7 \\ z_1 \end{Bmatrix}$	$\begin{Bmatrix} x_1 \\ x_2 \\ x_3 \\ y_5 \\ y_6 \\ y_7 \\ z_1 \\ z_3 \end{Bmatrix}$	$\begin{Bmatrix} x_1 \\ x_2 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ z_1 \\ z_2 \end{Bmatrix}$	$\begin{Bmatrix} x_1 \\ x_2 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ z_1 \\ z_2 \end{Bmatrix}$	$\begin{Bmatrix} x_1 \\ x_2 \\ x_3 \\ y_5 \\ y_6 \\ y_7 \\ z_1 \end{Bmatrix}$	$\begin{Bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ y_5 \\ y_6 \\ y_7 \\ z_1 \end{Bmatrix}$

The alternatives 1,4, and 5 have the same design variables and are regrouped as one architecture only (red columns). Same goes for alternatives 2 and 6 (blue columns).

This first step hence helps in reducing the number of architectures which will be used for the construction of the morphological tree, second step of the proposed methodology. An example implementation of morphological reduction is the software ENVISAGE [173].

5.1.2 Step 2: morphological tree

In order to accommodate the evolving size of the design space based on the configurations chosen at the macroscopic level, a tree structure is implemented to keep

track of the different design choices. As presented in section 2.3.1.2 page 141, the root of the tree is the morphological matrix of the macroscopic level while the leaves represent the morphological matrices of the individual agents. Model morphological matrices for the different architectures considered are stored at an abstract and intermediate level. These are used as templates for the leaves. This choice is inspired by object-oriented programming and the concepts of interfaces, abstract classes, and instantiation. This has been a tendency in systems engineering tools as well over the past few years with the examples of the Systems Modeling Language (SysML) and the Architecture Analysis and Design Language (AADL) using instance models of high-level interfaces. In the case of hardware specifications, this enables to have several available models or components to realize a given function. In terms of software, several algorithms can be instantiated to implement a specified task. Another advantage of such representations is that they can be used for analyses on a system, a classic example being a weight analysis where the analysis simply sums up the weight attributes of all subsystems instances. A representation of a morphological tree is given again here below (see Figure 5.4).

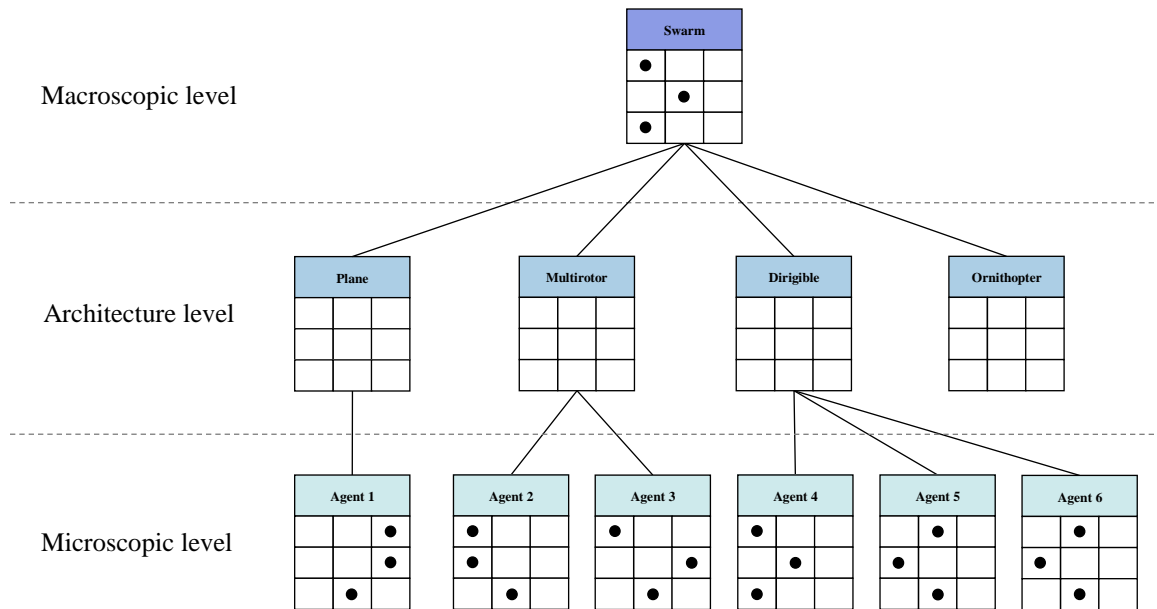


Figure 5.4: Example of morphological tree

Note that this representation can also be adapted when more than two levels are considered. In this case, abstract morphological matrices must be implemented at each new level so that each additional level in the design spaces translates into two additional levels in the morphological tree (one for the abstract morphological matrices, and one for the instantiations). In particular, the tree need not be balanced if subsystems exist for some architectures but not for others. Moreover, some options of morphological matrices might offer a variety of additional design choices. Given that such choices can be represented by a fixed morphological matrix, there is no need for a template. Hence, there is a distinction to be made between options which might be duplicated (one example is vehicle architectures as on Figure 5.4), and options which are chosen only once. These latter options are regrouped in what are called sublevels and are shown on Figure 5.5.

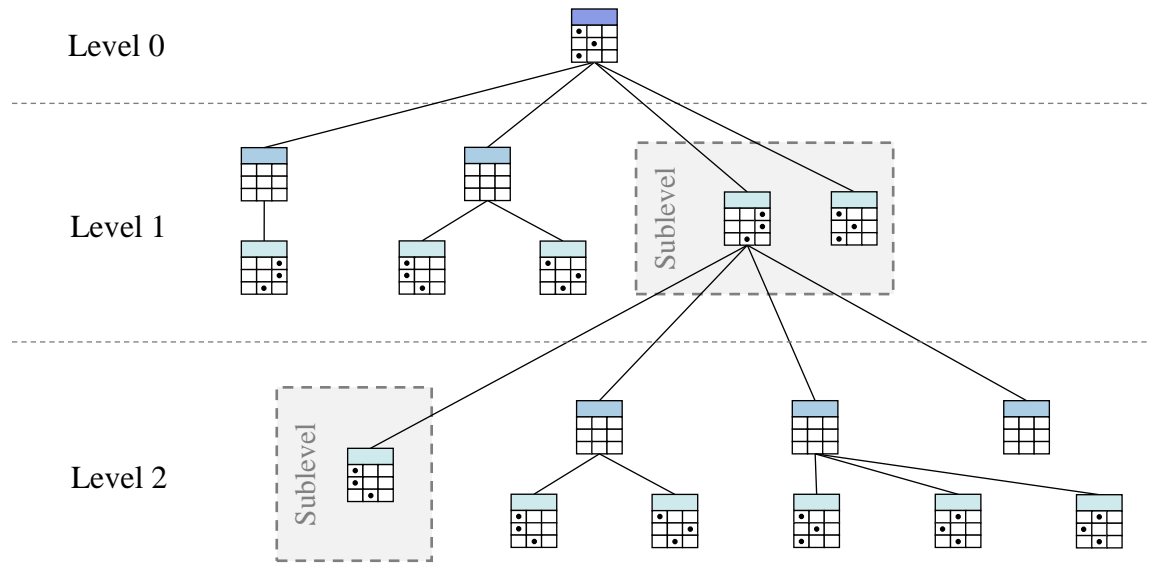


Figure 5.5: Multi-level morphological tree

To account for the feasibility of the generated alternatives, the tree structure must be coupled with compatibility matrices for each of the options (see section 5.1.1).

5.1.3 Implementation

The proposed example implementation of the morphological tree is mostly based on the object-oriented programming representation due to its flexibility: a key asset for dynamic design spaces. In particular, two main objects are to be implemented: a tree data structure, and a morphological matrix data structure.

5.1.3.1 Data structures

Firstly, the tree data structure is a widely used one and is available in many third party libraries. For the chosen Matlab implementation for instance, it is possible to find a number of open source files representing the tree data structure. The implementation of Jean-Yves Tinevez is chosen as it is recognized on the MathWorks repositories, has been downloaded and used many times by the community, and has received excellent ratings

[231]. This data structure is implemented as a class and includes many utility functions to perform operations on the tree. It is based on plain Matlab arrays and cell arrays and exhibits little overhead. Moreover, this tree data structure is a “per value” class as opposed to a “per reference” implementation, this has to be taken into account when modifying the tree structure. Finally, this implementation is modular enough so that any data type can be used for the roots and leaves of the tree.

To support the additional operations required on a tree of morphological matrices, a class *morphologicalTree* is created to inherit from the *tree* class. Its main additional functions are to perform reduction, the computation of the alternatives, and a string representation. Then, a class is created for morphological matrices and corresponding object instances are used in the tree structure. The *morphologicalMatrix* class is created as part of this research and the data structure is presented on Figure 5.6 using UML formalism.

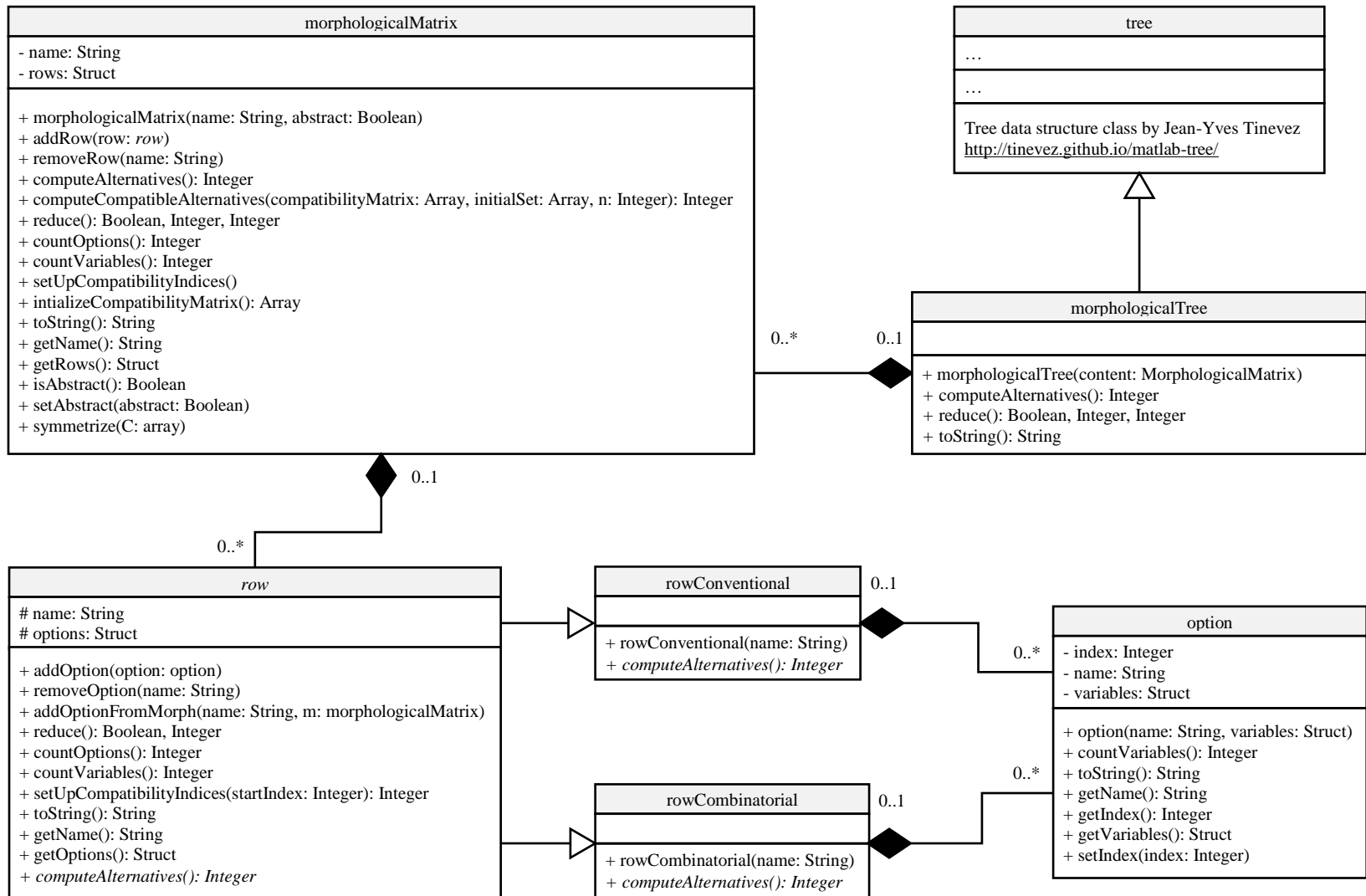


Figure 5.6: Proposed UML class diagram for the morphological tree

The morphological matrix basically consists of rows containing options. Each element is assigned a name which serves as a unique reference to each individual object. The options additionally comprise of a set of associated design variables which are used to perform the morphological reduction. The implementation contains basic operations to add and remove rows (features) and columns (options), and also carry out morphological reduction on rows and morphological matrices. For the rows, options containing the same variables are reduced to one, and for the morphological matrices, each row is reduced before removing rows containing only one option. The reduction methods return a Boolean indicating whether the structure was reduced or not, as well as the number of variables removed. Structures containing more than one object of a given type (matrix of rows, rows of options, options with several variables) are implemented as cell arrays in Matlab. Required constructors, getters, and setters are also employed.

Two types of rows are considered for the morphological matrix: conventional and combinatorial. The conventional row is a row where only one option can be chosen whereas several options can be chosen simultaneously in a combinatorial row. This latter type of row is particularly useful when combining sensors for instance (see Table 5.1 page 295). For conventional rows, the number of possible alternatives is just the number of elements in the options array whereas for the combinatorial row, all possible combinations have to be summed up for all numbers of options grouped together (Equation 5.3).

Equation 5.3: Total number of alternatives for a combinatorial row

$$n_{Alternatives} = \sum_{i=1}^{n_{Options}} \binom{n_{Options}}{i}$$

A row containing 3 options can be used as an example: if one option has to be chosen then $\binom{3}{1} = 3$ alternatives are possible, a number which corresponds to the conventional case. If two options are to be chosen, $\binom{3}{2} = 3$ alternatives can be considered. Finally, if the choice has to be made between 3 options, only $\binom{3}{3} = 1$ alternative is feasible. By summing all possibilities, one ends up with $3 + 3 + 1 = 7$ total alternatives for a combinatorial row of 3 options.

More in depth, this particular detail is implemented thanks to the concept of abstraction: an abstract class *row* is implemented with the abstract method *computeAlternatives*. This method is then detailed differently depending whether a conventional row or a combinatorial row is instantiated. Hence, two classes *rowConventional* and *rowCombinatorial* inherit from the abstract *row* class. To simplify the compatibility analysis, it is assumed that options from a combinatorial row are compatible within themselves. If this is not the case, one can always generate a conventional row from a combinatorial row by considering the required compatibilities (see Table 5.1 page 295).

Additionally, for the data structures to remain modular, a special method is implemented in the abstract *row* class which enables to create a full set of options from a

given morphological matrix. This is a key feature as it facilitates the multi-level morphological analysis. Considering the morphological interfaces shown in Table 5.3, Table 5.4, Table 5.5, Table 5.6, Table 5.7, and Table 5.8, this corresponds to the red cells which are individual options actually hiding full morphological matrices.

5.1.3.2 Morphological interfaces

Finally, before instantiating the tree of morphological matrices, the abstract morphological matrices of the represented architectures have to be constructed. Based on functional and physical decomposition (see Figure 3.13 page 207) of common models found in the commercial and military worlds, design features and their corresponding options are listed for each considered architecture. These example morphological interfaces are listed in this section and later used in the methodology to implement instances of each architecture.

Note: these matrices do not have the pretention to be exhaustive in any way, nor up to the standards used in the industry today where morphological matrices often comprise of thousands of rows. Rather, they serve as representative examples used to estimate the benefits of the morphological tree approach.

First, the most common type of UAV architecture is presented: the fixed-wing architecture. As it is shown on Table 5.3 and Table 5.4, multi-levels are present with the empennage, battery types, and landing gear wheel arrangements. These are represented as red options. Hence, by looking at this concrete example of morphological decomposition, one understands the need for a modular structure which can handle multiple levels such as

the morphological tree. Additional morphological decompositions follow for the macroscopic level as well as for the multicopter, airship, and ornithopter architectures.

Table 5.3: Example of empennage morphological matrix

Features	Options				
Number of tailplanes	0 • Tailless • Canard	1	2	3	
Location	Low	Mid	High	Booms • Tail • Wing	
Moving surfaces	Independent	Stabilator			
Number of fins	0	1	2		
Configuration	Fin/Taiplane	V tail	Inverted V tail	X tail	Pelikan
750 alternatives					

Table 5.4: Fixed-wing architecture morphological interface



Fixed-Wing

Features		Options				
Aerodynamics	Body	Separate fuselage	Flying wing	Blended body	Lifting body	
	Wing	Straight	Swept	Delta	Compound delta	
	Wing position	Low	Mid	Shoulder	High	Parasol
	Detachable wing	Yes	No			
	Empennage	None	See Table 5.3			
	Type of launch	Horizontal	Vertical	Hand-launched	Aircraft-launched	Catapult-launched
Type of landing	Horizontal landing	Vertical landing	Energy dissipation crash	Parachute	Net	
Propulsion	Number of motors	1	2	3	4	
	Energy source	Bio-chemical	Electric charge	Solar	Electrolyte	Hybrid
	Energy storage	Battery • LiCoO2 • LiFePO4 • LiPo • NiCad • NiMH	Fuel tank	External fuel tank	Electrolyte tank Fuel cell	Fuel tank + Battery • LiCoO2 • LiFePO4 • LiPo • NiCad • NiMH External fuel tank + Battery • LiCoO2 • LiFePO4 • LiPo • NiCad • NiMH
	Converter to mechanical energy	Piston	Turbine	Electric motor	Hybrid (Piston/Electric)	Hybrid (Turbine/Electric) Fuel cell and electric motor
	Converter to lift/thrust	Rotor	Fan	Propeller	Jet	
Sensors	Imaging	Mono • Fixed-mount • Gyro-stabilized	RGB camera • Fixed-mount • Gyro-stabilized	Multispectral data • Fixed-mount • Gyro-stabilized	Thermal camera • Fixed-mount • Gyro-stabilized	
	Mapping	2D LIDAR	3D LIDAR	Sonar		
	Attitude	IMU+GPS				
	Altitude	GPS	Barometer	Sonar		
	Communications	Radio				
Structures	Landing gear arrangement	None	Wheels • Tail wheel • Tandem • Tricycle • Wing	Skids	Floaters	Skis
	Landing gear type	None	Fixed	Retractable		
	Landing gear shock absorption	Rigid	Leaf-type	Bungee cord	Shock struts	

Table 5.5: Multirotor architecture morphological interface



Features		Options					
Aerodynamics	Fairings	None	Electronics only	Full body	Full body and payload		
	Type of landing	Vertical landing	Energy dissipation crash	Parachute	Net		
Propulsion	Energy source	Bio-chemical	Electric charge	Electrolyte	Hybrid		
	Energy storage	Battery • LiCoO ₂ • LiFePO ₄ • LiPo • NiCad • NiMH	Fuel tank	External fuel tank	Electrolyte tank Fuel cell	Fuel tank + Battery • LiCoO ₂ • LiFePO ₄ • LiPo • NiCad • NiMH	External fuel tank + Battery • LiCoO ₂ • LiFePO ₄ • LiPo • NiCad • NiMH
	Converter to mechanical energy	Piston	Turbine	Electric motor • Spinning cage • Spinning shaft • Brushed • Brushless	Hybrid (Piston/Electric)	Hybrid (Turbine/Electric)	Fuel cell and electric motor
	Rotorcraft	Single rotor	Coaxial	Tail sitter	Tilt-rotor	Multirotor	
	Number of rotors	1	2	3	4	6	8
	Converter to lift/thrust	Rotor	Fan	Propeller			
	Number of blades	2	3	4	6	8	10
	Rotors/Frame arrangement	I	X	Y	V		
	Blade type	Fixed pitch	Variable pitch				

Table 5.5 (continued)

Sensors	Imaging	Mono • Fixed-mount • Gyro-stabilized	RGB camera • Fixed-mount • Gyro-stabilized	Multispectral data • Fixed-mount • Gyro-stabilized	Thermal camera • Fixed-mount • Gyro-stabilized	
	Mapping	2D LIDAR	3D LIDAR	Sonar		
	Attitude	IMU+GPS				
	Altitude	GPS	Barometer	Sonar		
	Communications	Radio				
Structures	Landing gear arrangement	None	Foam pads	Skids	Floaters	
	Landing gear type	None	Fixed	Retractable		
	Landing gear shock absorption	None	Rigid	Leaf-type	Bungee cord	Shock struts
	Frame type	Aerial cinematography	Sport	Sport FPV	Mini	Mini FPV

Table 5.6: Macroscopic level morphological interface

Features		Options				
Mission type	HALE	Long-range strike	MALE	Close-range support	MUAV	MAV
Architecture	Fixed/Conventional	Product family	Scale-based product family	Reconfigurable	Online reconfigurable	Modular
Control type	Centralized	Decentralized				
Control scheme	Leader/Follower	Hierarchical	Consensus	Distributed		
Ground station	Remote base	Laptop	Wearable technology			

Table 5.7: Morphological interface for the airship architecture



 Airship							
Features			Options				
Aerodynamics	Lifting medium	Cold gas (Helium)	Hot air				
	Empennage configuration	Y	Inverted Y	X	Cross		
	Ballonet-based pitch trim	Yes	No				
Propulsion	Energy source	Bio-chemical Battery	Electric charge	Electrolyte	Hybrid		
	Energy storage	<ul style="list-style-type: none"> • LiCoO2 • LiFePO4 • LiPo • NiCad • NiMH 	Fuel tank	External fuel tank	Electrolyte tank Fuel cell	Fuel tank + Battery <ul style="list-style-type: none"> • LiCoO2 • LiFePO4 • LiPo • NiCad • NiMH 	External fuel tank + Battery <ul style="list-style-type: none"> • LiCoO2 • LiFePO4 • LiPo • NiCad • NiMH
	Converter to mechanical energy	Piston	Turbine	Electric motor <ul style="list-style-type: none"> • Spinning cage • Spinning shaft • Brushed • Brushless 	Hybrid (Piston/Electric)	Hybrid (Turbine/Electric)	Fuel cell and electric motor
	Number of rotors	1	2	3	4	6	8
	Steerable propulsion Converter to lift/thrust	Yes	No				
	Number of blades	Rotor	Fan	Propeller			
	Blade type	2	3	4	6	8	10
Sensors	Imaging	Mono	RGB camera	Multispectral data	Thermal camera		
	Mapping	<ul style="list-style-type: none"> • Fixed-mount • Gyro-stabilized 	<ul style="list-style-type: none"> • Fixed-mount • Gyro-stabilized 	<ul style="list-style-type: none"> • Fixed-mount • Gyro-stabilized 	<ul style="list-style-type: none"> • Fixed-mount • Gyro-stabilized 		
	Attitude	2D LIDAR	3D LIDAR	Sonar			
	Altitude Communications	IMU+GPS GPS Radio	Barometer	Sonar			
Structures	Hull type	Non-rigid	Semi-rigid	Rigid			
	Battens	Yes	No				

Table 5.8: Morphological interface for the ornithopter architecture [232], [233]

 Ornithopter					
Features			Options		
Aerodynamics	Wing twisting	Yes <ul style="list-style-type: none"> • Spar rotation • Spar torsion • Servo-controlled • Auxiliary spar 	No		
	Wing type	Flying wing	Tandem wing	Thrust-wing	Oscillating stretched wing Rotating wing
Propulsion	Energy source	Bio-chemical	Electric charge	Other	
	Energy storage	Battery <ul style="list-style-type: none"> • LiCoO2 • LiFePO4 • LiPo • NiCad • NiMH 	Fuel tank	Rubber	
	Gearbox type	Strut	Plate		
	Gear type	Cluster	Spur with pinion wire		
	Flapping mechanism	Staggered crank	Outboard wing hinge	Dual cranks	Transverse shaft
	Converter to mechanical energy	Internal combustion engine	Electric motor <ul style="list-style-type: none"> • Spinning cage • Spinning shaft • Brushed • Brushless 	Rubber + shaft	
Sensors	Imaging	Mono	RGB camera	Multispectral data	
	Attitude	IMU+GPS			
	Altitude	GPS	Barometer	Sonar	
	Communications	Radio			
Structures	Wing reinforcement	Battens	Perimeter		

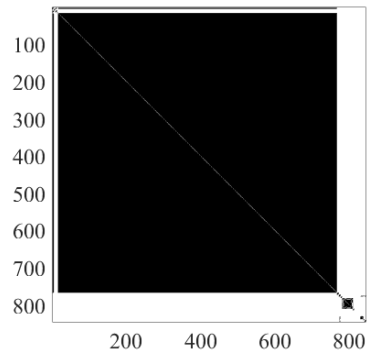
5.1.3.3 Compatibility study

In conjunction with the morphological interfaces, compatibility matrices are required to consider only alternatives which options are all compatible. First, indices are associated with every option of the morphological matrix thanks to the method *setUpCompatibilityIndices*. The numbering convention is shown in Table 5.9.

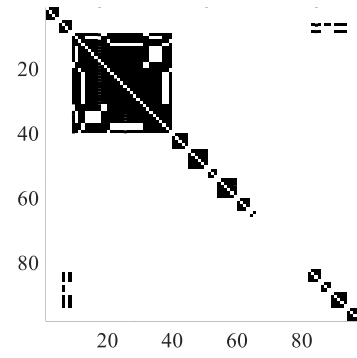
Table 5.9: Options compatibility indexing convention

Features	Option 1	Option 2	Option 3	Option 4
Feature 1	1	2	3	
Feature 2	4	5		
Feature 3	6	7	8	9
Feature 4	10	11		

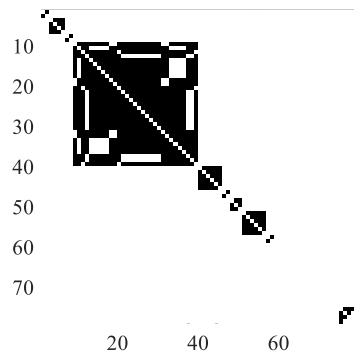
Then, the function *initializeCompatibilityMatrix* initiates a compatibility matrix assuming that all options are compatible and then prefills the incompatible trivial blocks as explained in Figure 5.2. The designer is then left to identify only the incompatibilities. For each architecture, the compatibility matrices can be visualized in Figure 5.7.



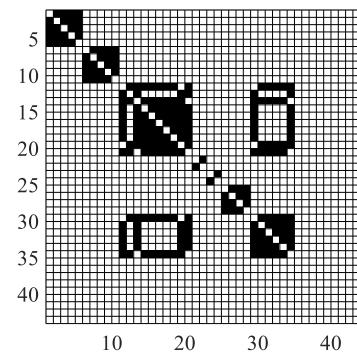
(a) Fixed-wing (845×845)



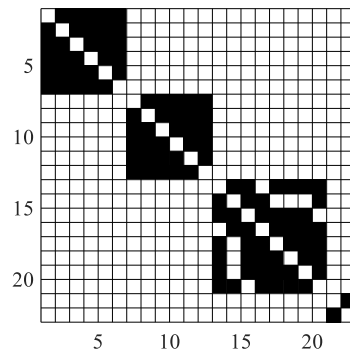
(b) Multirotor (97×97)



(c) Airship (79×79)



(d) Ornithopter (44×44)



(e) Macroscopic level (23×23)

Figure 5.7: Compatibility matrices

Once the compatibility matrices are prepared, the compatible alternatives must be generated. Given the extremely high number of total possible (and not necessarily compatible) alternatives, a full factorial approach eliminating incompatible options cannot be considered. Although the proposed morphological matrices remain small with respect to what is observed in the industry, generating the full factorial for all possible alternatives is impossible due to computer memory constraints. For example, generating all possible alternatives for the airship architecture would require 100 GB of memory. This full factorial array would then have to be run through to identify and rule out incompatible alternatives. As a consequence, a recursive approach on the rows of the morphological matrix is chosen as in the fashion of [173]. For a given function call, the algorithm tries to add each option of the current row to the considered alternative. If the option is compatible with the alternative built so far (step n), the option is added to the alternative and the recursive function is called on the following rows (step $n + 1$). The stopping condition of the recursion is reached when no more rows remain. This method is very similar to traveling a tree in a depth-first fashion (see Figure 5.9).

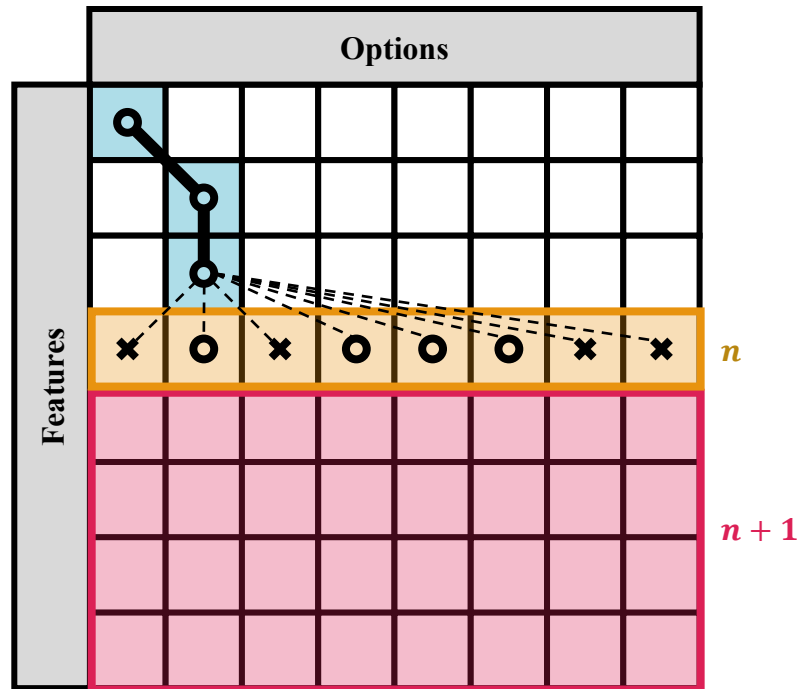


Figure 5.9: Recursive count of compatible alternatives

In this particular function call, three options have been chosen and locked in and the algorithm is looping over and considering all the options of row n . The circles represent options compatible with the current alternative (green options) and the crosses represent the incompatible ones. For each compatible option (circle), the function will have a recursive call over the sub-problem: the remaining rows of the morphological matrix (red area) and repeat the same process for row $n + 1$. The complete workflow for generating compatible alternatives is the following:

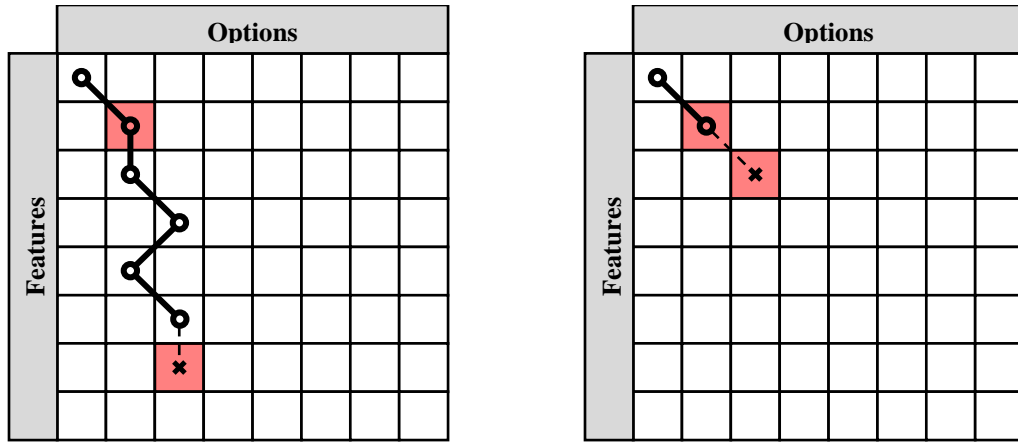
1. Prepare morphological matrix
2. Initialize options indices
3. Initialize compatibility matrix
4. Fill in known incompatibilities
5. Run recursive algorithm to count compatible alternatives

The complete compatibility analysis is summarized in Table 5.10. Due to the size of the morphological decomposition of the fixed-wing and multicopter architectures, compatibility studies were to computationally expensive to run.

Table 5.10: Number of compatible alternatives

Architecture Name	Combinatorial alternatives	Total incompatible pairs	Non-trivial Incompatible pairs	Compatible alternatives	Compression
Fixed-wing	3.8916×10^{16}	284,340	2,441	-	-
Multirotor	1.6789×10^{14}	500	214	-	-
Airship	3.3579×10^{11}	441	199	111,974,400	99.97%
Ornithopter	7,408,800	113	46	158,400	97.86%
Macroscopic level	1620	56	10	540	66.67%

The recursive function takes a couple hours to run for the small matrices and lists all possible compatible alternatives, making it a pretty long process. However, the procedure can be sped up by properly conditioning the problem. Indeed, the algorithm will take a long time if for many occasions the “depth first” search is able to go deep in the alternatives tree. Since the number of compatible alternatives is given by the problem, it is not possible to change the number of branches where the algorithm goes the deepest. However, it is possible to change how early the incompatibilities are found. The earlier an incompatibility is found, the longer the underlying branches are and the more possibilities are removed at once (see Figure 5.10).



(a) Badly conditioned problem: incompatibility found in the last recursive layers

(b) Good conditioning: incompatibility found from the beginning, preventing the proliferation of child branches

Figure 5.10: Problem conditioning

The options of the considered alternative which are incompatible together are highlighted in red. If the option incompatible with the option chosen at the first or second row is listed earlier (Figure 5.10 (a)), the algorithm will not pursue the search with another multitude of branches (Figure 5.10 (b)). Hence, the idea is to condition the morphological matrix so that most incompatibilities lie in the first rows of the matrix (since they are considered first by the algorithm). Thanks to this pre-conditioning of the problem, most incompatible alternatives are removed at the beginning of the recursive process and the algorithm is able to finish in a much shorter time. However, as can be seen in Table 5.10, the function call would take too long (respectively from around a hundred days to twenty-two years) for the multirotor and the fixed-wing architectures. Note that the compatibility matrix has to be formed once reduction has been performed or the reduction step has to also properly reformat the compatibility matrices.

5.1.3.4 Morphological reduction

Final step of the implementation, the morphological reduction is easily performed thanks to the object-oriented programming paradigm. Indeed, the reduction is first performed at a row level for each option of the row. Options which are found to have the same design variables as another are removed from the row. Then, the reduction continues at the morphological matrix level: once the row has been reduced and if there is only one option left, this latter is removed from the matrix. Finally, a *reduction* method can be implemented for a tree object and that method performs reduction for each of the nodes of the tree. Note that in the extreme case when a whole morphological node is reduced to one option, it could potentially be removed from the tree and directly transferred to the optimization step as it is done for rows with single options.

Going back to the example morphological matrices introduced in section 5.1.3.2 page 306 and analyzed in Table 5.10 page 318, it is possible to perform reduction and obtain a first glance at the type of results obtainable through morphological reduction (Table 5.11 and Figure 5.11).

Table 5.11: Number of reduced alternatives

Architecture Name	Initial combinatorial alternatives	Reduced combinatorial alternatives	Options removed	Non-unique variables removed	Compression
Fixed-wing	3.8916×10^{16}	1.6463×10^{13}	712	4752	99.96%
Multirotor	1.6789×10^{14}	6.4774×10^{11}	25	37	99.61%
Airship	3.3579×10^{11}	3.2387×10^8	27	50	99.90%
Ornithopter	7,408,800	3,175,200	4	4	57.14%
Macroscopic level	1620	1620	0	0	0%



Figure 5.11: Effect of options removal on overall reduction

Observing these results, the macroscopic level morphological analysis is not reduced. This is due to the fact that the options available at the macroscopic level are very particular in the proposed case and all have very unique design variables. Note that this might be different for other canonical examples. Then, reduction seems all the more important when the morphological decomposition is large. One example is the fixed-wing architecture which encompasses a secondary morphological decomposition for the empennage, adding more than 750 new alternatives multiplying the other analysis. For this architecture, the compression due to reduction is more than 99.95% with a very large number of options removed (mainly from the empennage decomposition). However, with the multirotor and airship configurations, a high reduction is achieved (more than 99%) even though the proportion of options removed is reasonable (25% to 34%). Hence, reduction seems to show promising results in reducing the size of the design space even when not many options are removed. For the ornithopter architecture, removing only 4

options results in a reduction of the design space of more than 50%. A higher reduction is not achieved as the options for this type of peculiar architecture are very unique and not standardized.

Propagating these results to the full morphological tree, a tremendous reduction of the design space is achieved, demonstrating the multiplied power of morphological reduction when applied to multi-level design spaces (Figure 5.12).

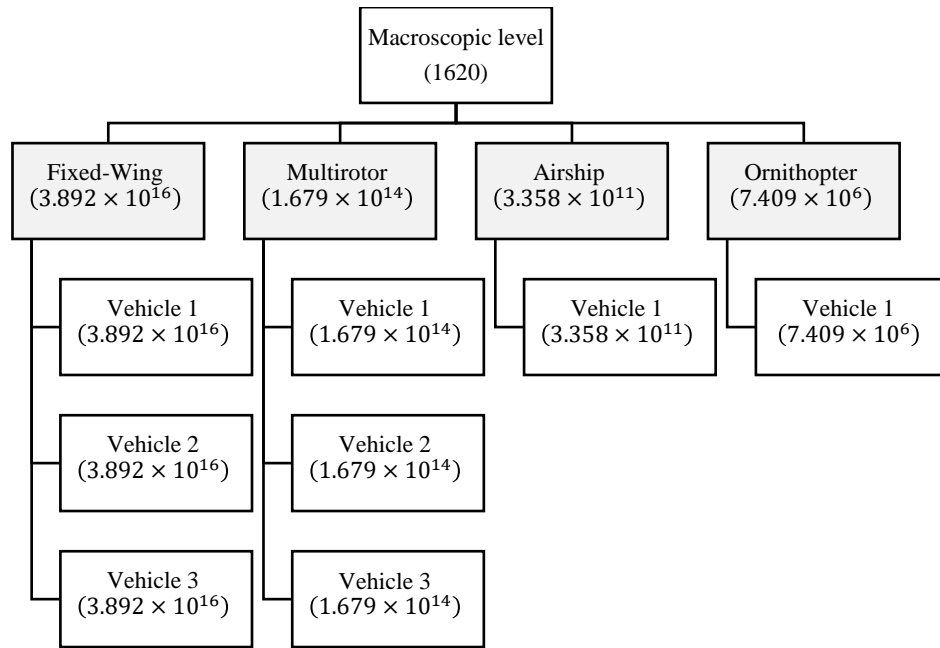


Figure 5.12: Initial morphological tree before reduction

The abstract level is represented with the gray boxes and these are not accounted for to compute the total number of alternatives in the tree. Before reduction, the tree comprises of 1.1242×10^{14} combinatorial alternatives. After reduction, this number goes down to 2.0203×10^{93} by removing 3010 options (around 20,000 non-unique variables).

Even though the number of alternatives to be considered is still enormous with respect to current design exploration techniques, the reduction achieved is quite important and this number of alternatives has been reduced by several orders of magnitude.

5.1.4 Verification and validation

The verification and validation of the morphological tree with reduction is simply performed through a campaign of unit testing. Moreover, this section refers to the validation criteria of experiment 3.1 defined in chapter 2 and validates hypothesis 3.1.

Are all alternatives feasible? The complementary approach of the compatibility matrix ensures that feasibility is enforced for each matrix (leaf) of the morphological tree but also between the different levels when alternatives are generated. Hence all alternatives provided by the morphological tree are feasible. The unit test associated with this criterion generates random alternatives and for each one of them, looks up the chosen technologies in the compatibility matrix. The test passes only if no incompatibility is found.

Are there still redundant variables or options groups in the resulting alternatives?

This step ensures that the morphological reduction step is properly carried out and propagated through the different levels of the tree. The corresponding unit test performs morphological reduction on a randomly generated design space. Then, it reduces the resulting design space a second time and compares whether there was a second size change or not. The test passes only if the size of both design spaces is the same, which indicates that the first reduced design space is no longer compressible.

Is the number of alternatives reduced when compared with the classical morphological approach? This question is easily answered by generating a random swarm constitution and computing the resulting number of alternatives from this design space. As a second step, morphological reduction is performed and the total number of alternatives are compared in both cases.

Are all existing concepts covered by the generated alternatives? The morphological approach ensures that every option of the tree is considered in the alternatives. Indeed, the generation of alternatives is done with a full factorial decomposition of all possible options, hence certifying that any given option is used in the alternatives. Covering all existing concepts then depends on the precision of the literature review and the definition of the morphological interfaces. In the example implementation, notional morphological matrices are provided and most likely lack some options in their decomposition while still providing a wide coverage of the design space for unmanned vehicles.

Can the generated alternatives be easily fed to the optimization and analysis algorithms? Each alternative generated by the morphological tree comes with a set of design variables used to define it. Using this architecture in the optimizers (such as the one defined in the next section) then depends on implementing the behavior of the architecture as a function of its design variables, and defining bounds for each design variable. This step is relatively easy since:

- The number of architectures to be implemented is most likely reduced thanks to morphological reduction.

- The morphological tree directly provides the design variables to be used in the implementation.

5.1.5 Characterization

This section provides a complete characterization of the proposed design space exploration technique. Both aspects are considered with a first part focusing on morphological reduction and a second part detailing the morphological tree characterization.

5.1.5.1 Morphological reduction

An efficient design space exploration technique relies on two main capabilities: covering a maximum of concepts and simultaneously reducing the execution time required to reach optimal solutions. Although the usage of the morphological approach guarantees the exhaustiveness of the generated concepts with respect to the listed options, it is not granted that the computation time will remain reasonable. Indeed, two aspects have to be considered. On one hand, the number of discrete cases to be run in the optimizers has been reduced thanks to morphological reduction. On the other hand, a certain number of corresponding design variables has been transferred to the optimizer. In order to conclude on the effectiveness of the whole methodology, it is critical to study whether the morphological reduction is outbalanced by the number of additional design variables in the optimization process. Introduced by [173], this concept named “morphological reduction” in this work is applied to multi-level design spaces in this section and further characterized.

A first step in this approach is to assess the increase in computation time incurred by the multiplication of design variables used by the optimizer. In order to be consistent

with the optimizer used in this research, a genetic algorithm optimizer is used on a set of thirteen multivariable test functions in the same fashion as [234]. These optimization test functions have the particularity to have an adjustable number n of design variables, so as to study the behavior of the optimization algorithm with respect to n : key requirement for the present test. Such functions are presented in Appendix A.2 page 432.

For a given number of design variables, this analysis runs the genetic algorithm to optimize the test function. In particular, the same stopping criteria of stability and stalling generations is used for the convergence criteria. Such runs are replicated over a hundred times to ensure a proper statistical analysis of the results as well as robust conclusions (Figure 5.13).

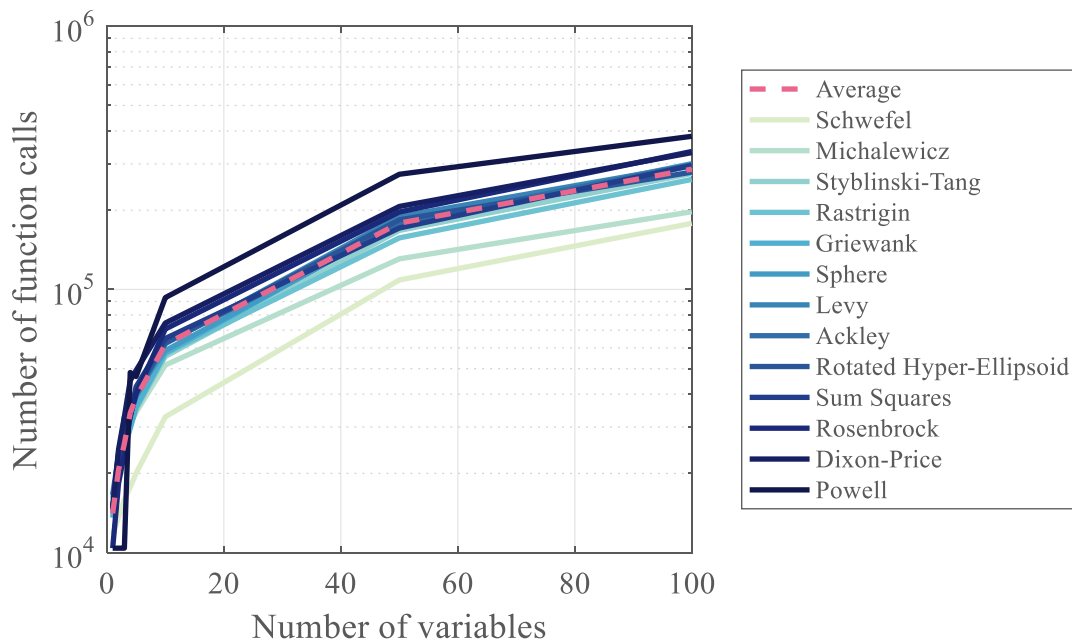


Figure 5.13: Number of function calls versus number of design variables

The number of function calls required for convergence quickly shoots up between 1 and 20 design variables, before increasing more slowly. Although there is no apparent limit to the number of function calls, the tendency of the curves reminds of an asymptotic behavior. In order to obtain a general trend of how the number of function calls in the optimizer is affected by the number of design variables, the average number of function calls is computed for every number of design variables (see Table 5.12).

Table 5.12: Average number of function calls for the test functions

Number of design variables	Average number of function calls
1	14,121
2	20,890
3	26,553
4	34,637
5	40,145
10	71,679
50	237,322
100	408,899

The average number of function calls required for convergence can then be approximated by Equation 5.4. With n the total number of design variables used by the optimizer. The coefficient of determination for this fit is $R^2 = 0.9992$.

Equation 5.4: Surrogate model for the number of function calls

$$f_c(n) = -11.776n^2 + 5,127.9n + 13,236$$

In order to fully understand the effects of morphological reduction, the analysis is separated between single level and multilevel effects.

5.1.5.1.1 Single level

This subsection draws out new conclusions (with respect to [173]) on the performance of morphological reduction. For all following examples, a notional morphological matrix is considered with 8 rows and 4 options per row. The initial number of variables in the optimizer is 50. Options are then removed one by one with morphological reduction and the influence of different parameters is studied. It is assumed for the following figures that options are removed row by row, as opposed to column by column. However, the effect of both strategies is studied on Figure 5.20 and Figure 5.21.

In Figure 5.14, the number of design variables removed per option (also denoted as factor k) is 4. The nominal case without morphological reduction is represented in blue and does not change as options are removed.

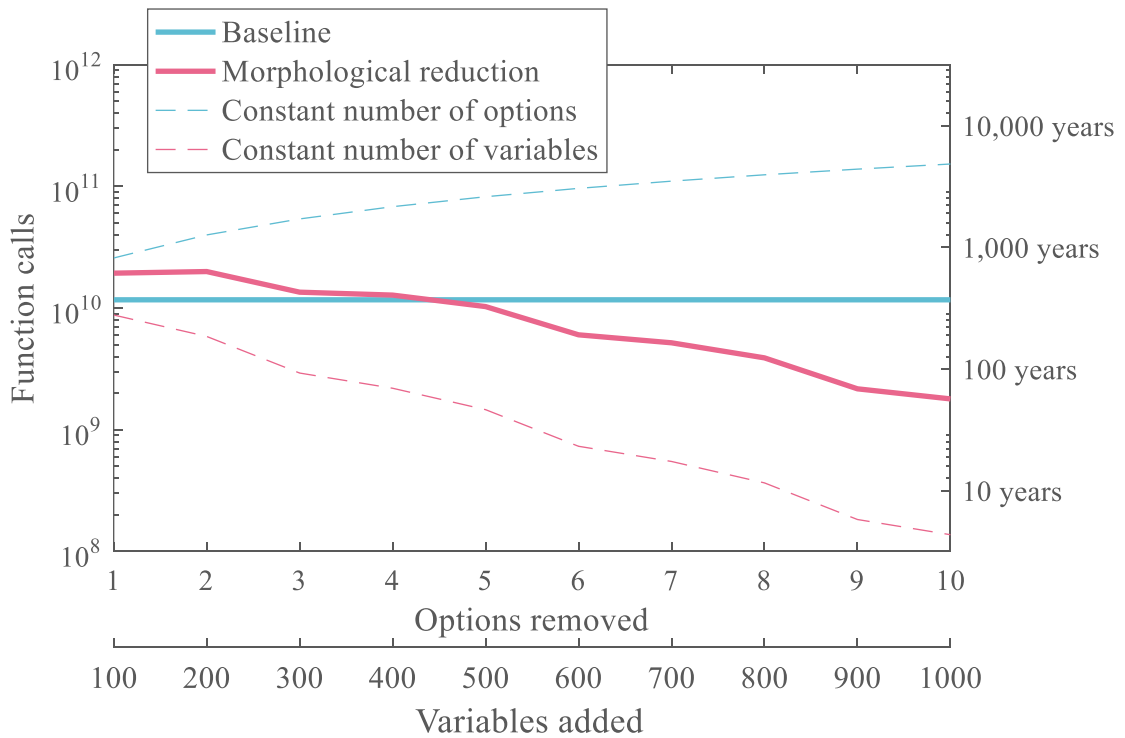


Figure 5.14: General effect of morphological reduction

A first observation is that the morphological reduction curve (in solid red) is always below the nominal case (solid blue): the number of total function calls is always lower with morphological reduction. The vertical right axis represents an equivalent duration if each function call took one second to evaluate. It seems that the principal influence comes from the reduction in the number of discrete cases in the morphological decomposition, and that the resulting increase in the number of design variables for the optimizer has a minor contribution in the number of function calls. Indeed, one possible way to check this statement is to compute the number of function calls when design variables are not transferred to the optimizer: options (and variables) are simply removed. This curve is shown in dashed red and appears only slightly below the solid red curve. This confirms that the action of transferring design variables to the optimizer as options are removed has only a minor influence on the number of function calls. The same conclusion is drawn when considering the complementary approach: keeping the number of discrete calls constant and increasing only the number of design variables in the optimizer (dashed blue curve). Again, the difference with the baseline is smaller than with the solid red line. As a conclusion, the influence on the number of function calls is dominated by the reduction in the number of times the optimizer has to be run and not by its number of variables.

Although the first example on the baseline problem seems to show that morphological reduction is always advantageous, it is essential to reach the limits of this experiment and see if it is possible to obtain a case where such a clear conclusion cannot be drawn. By pushing the factor k to 100 (100 variables added per option removed), Figure 5.15 is obtained.

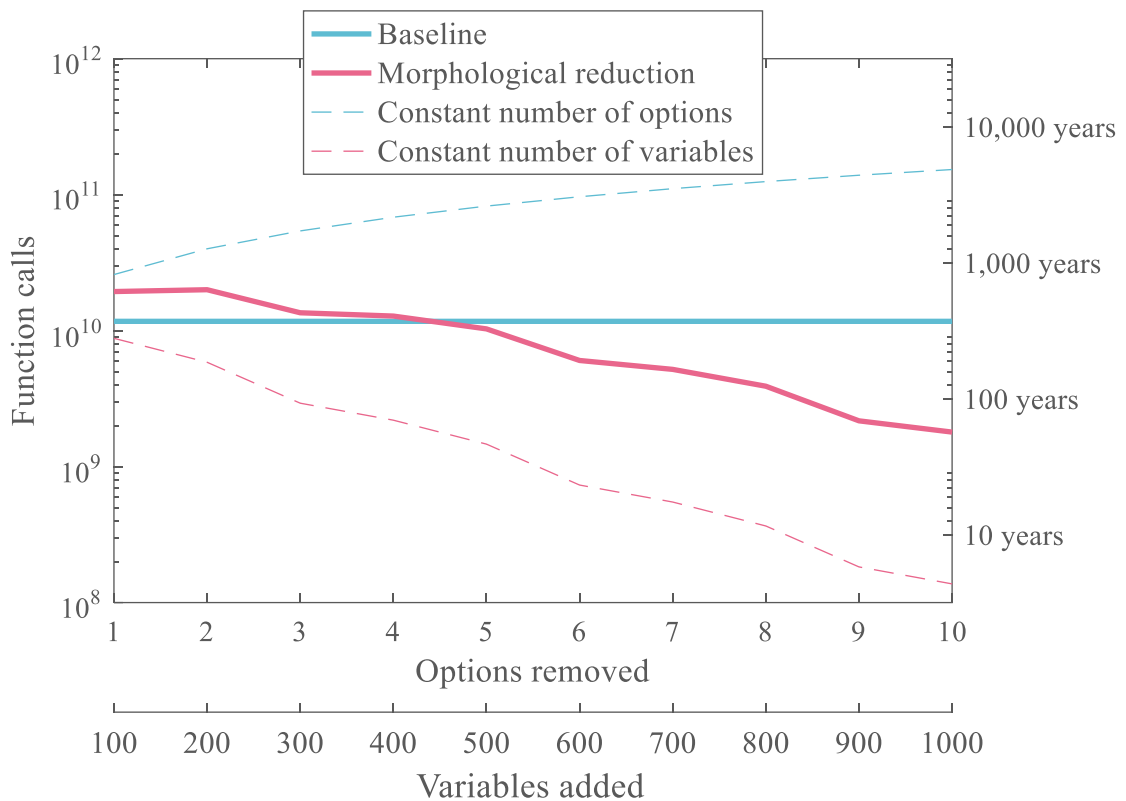


Figure 5.15: Limits of morphological reduction

The red and blue solid curves now intersect and there seem to be a tradeoff between the number of times the optimization algorithm is run, and the number of design variables transferred to it. When 1 to 4 options are removed from the morphological matrix, this means that 100 to 400 additional variables are transferred to the optimizer. This ends up dominating the number of function calls. Hence, it is more computationally expensive in this case to use morphological reduction. As for the previous example, this trend is confirmed by looking at the dashed curves and their distance from their respective solid counterparts. By the time 10 options have been removed, the variation between the red solid and dashed curve is the same as the total variation of the solid line: adding variables to the optimizer now has the same impact as the number of calls to the optimizer. To

conclude on this example, it seems that in some extreme cases, the proposed morphological reduction might take longer and be more expensive than the conventional approach. However, this was derived for the unusual case when one option of the morphological matrix requires 100 variables to be represented, a number much larger than the initial 50 variables of the baseline problem.

The effect of this parameter k is further studied on Figure 5.16. It is varied from 2 to 4 with the same number of initial design variables in the optimizer.

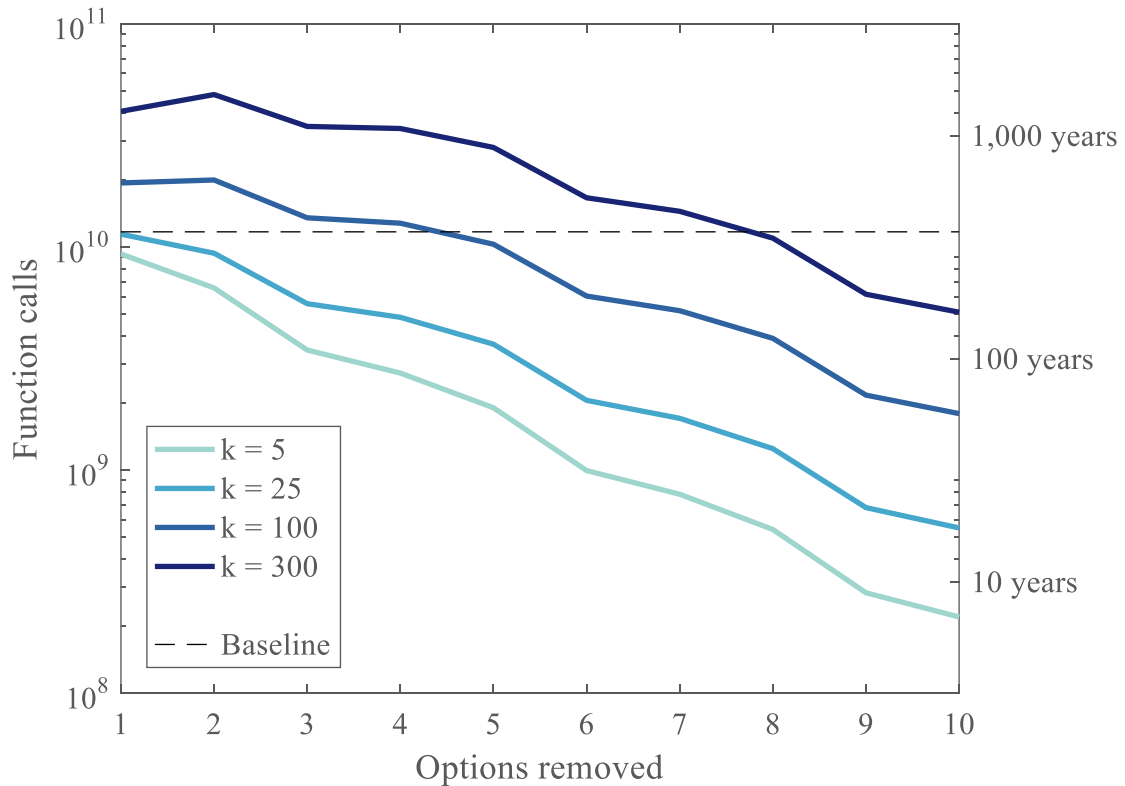


Figure 5.16: Influence of factor k on morphological reduction

As k is increased (more variables transferred per option removed), morphological reduction requires more and more function calls and gets closer to the nominal case. In the extreme case when $k = 300$ variables are transferred to the optimizer per option removed, the number of function calls increases from 1 to 2 options removed. This is opposed to the previous observations when the trend was always decreasing for the morphological reduction curves. This can also be represented by a contour plot listing the zones where the morphological reduction uses more or less function calls than the full morphological decomposition (Figure 5.17).

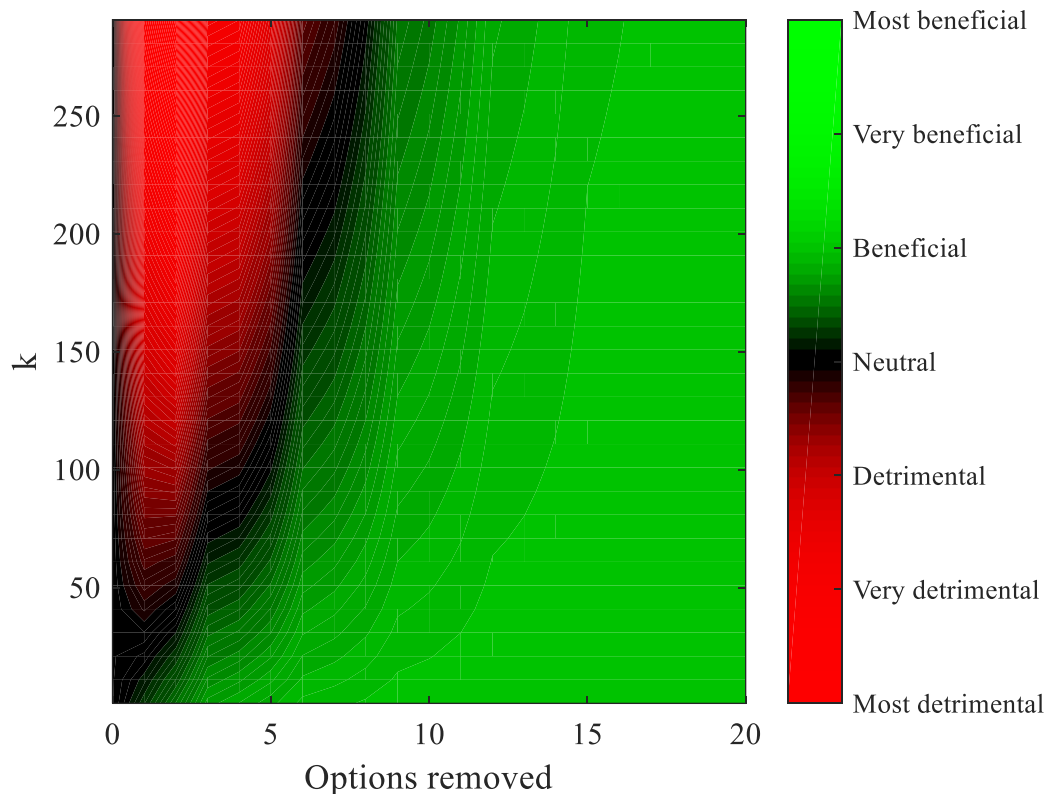


Figure 5.17: Contours of morphological reduction profitability

This map shows that even for a small morphological matrix example, the zone where morphological reduction is detrimental remains quite limited in terms of coverage. Even if the k-factor varies from 1 to 300 variables added per option removed, there is no detrimental zone after 8 options have been removed from the morphological matrix.

Finally, the repercussions of the size of the problem are considered in these last few examples. First, the number of rows in the morphological matrix is varied in Figure 5.18.

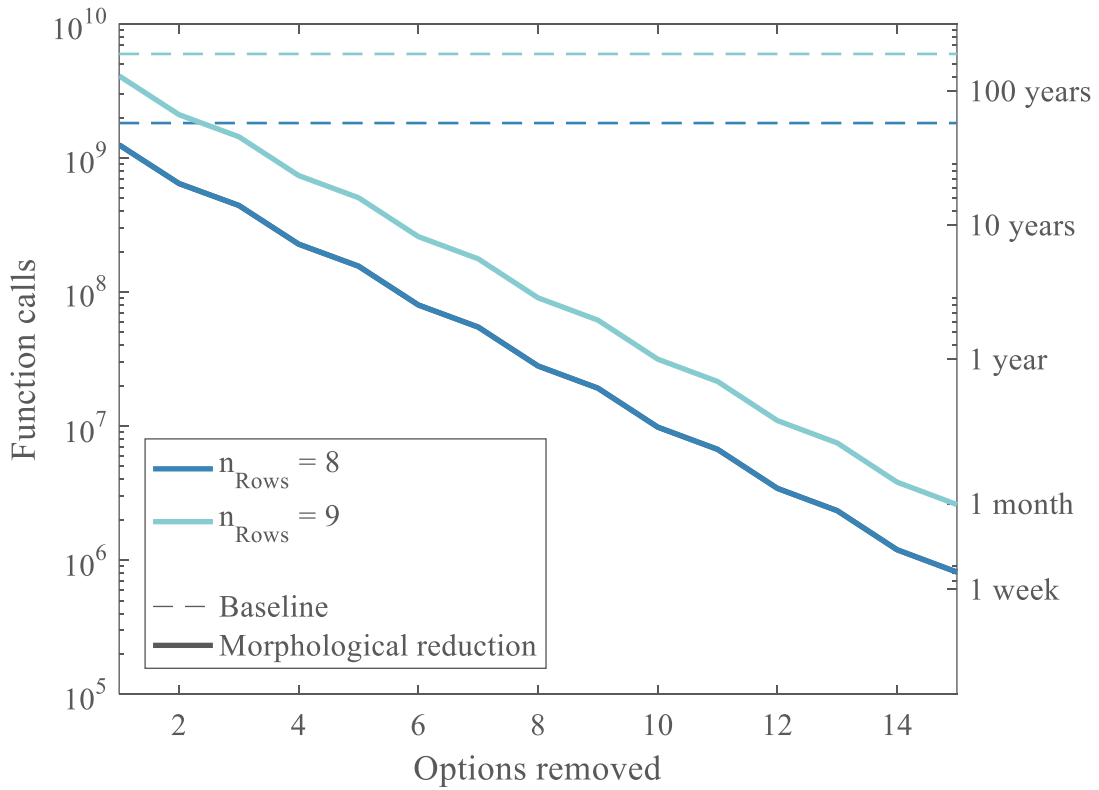


Figure 5.18: Influence of the number of rows on morphological reduction

As expected, with more rows in the morphological matrix, the computational time increases for both the baseline and the morphological reduction cases. With both problem sizes, the morphological reduction still performs better than the classic approach. The same observations are drawn from Figure 5.19 when the number of options per row is varied.

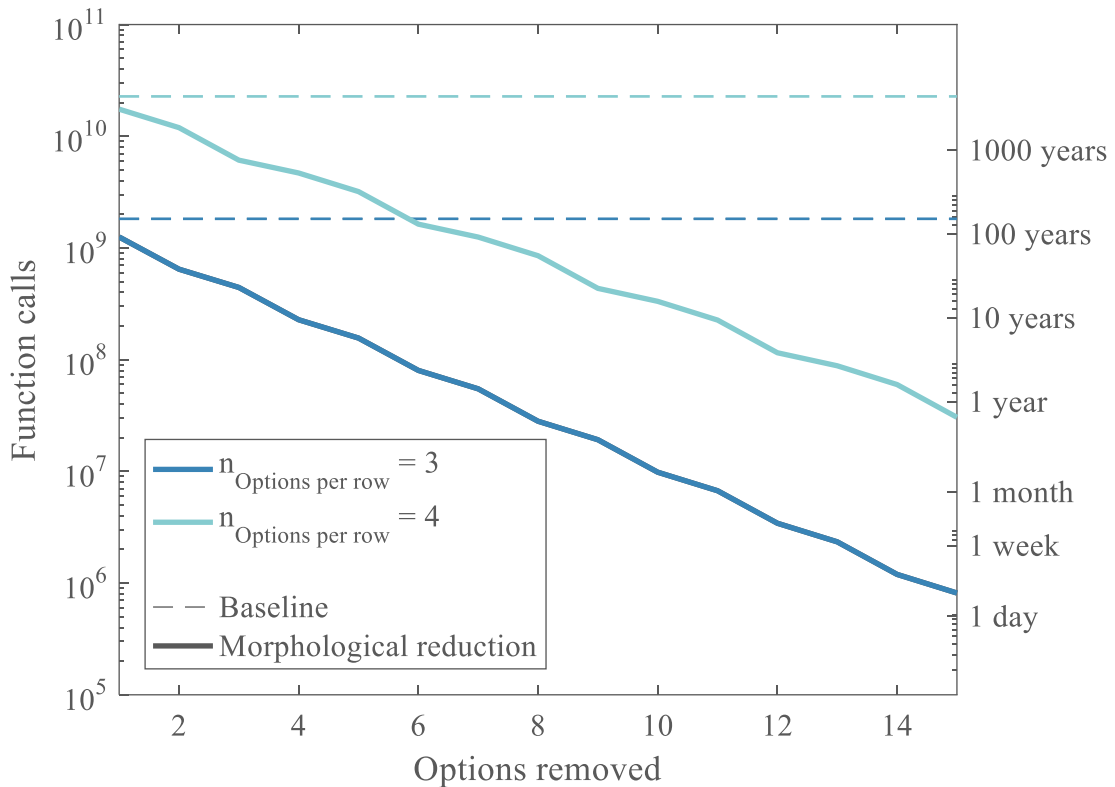


Figure 5.19: Influence of the number of options per row on morphological reduction

It can be observed that the number of options per row has more influence on the response than the number of rows in the morphological matrix. This is reminiscent of the strategy chosen for the elimination of options which was mentioned at the beginning of this section. Two strategies were envisaged: rows first or columns first. Figure 5.20

demonstrates both strategies on the notional morphological matrix of this section by removing 10 options. Removed options are represented in red.

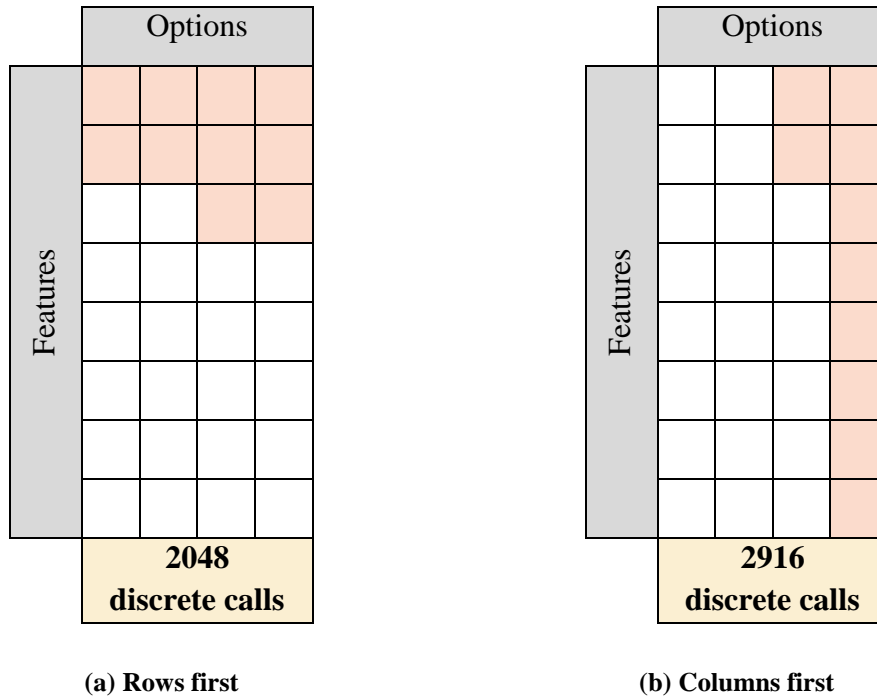


Figure 5.20: Strategies of options removal

The “rows first” strategy leaves 2048 possible alternatives (i.e. 2048 calls to the optimizer) while the columns first leaves 2916 of them. Given this important difference, it is essential to consider both testing strategies for the characterization of morphological reduction (see Figure 5.21).

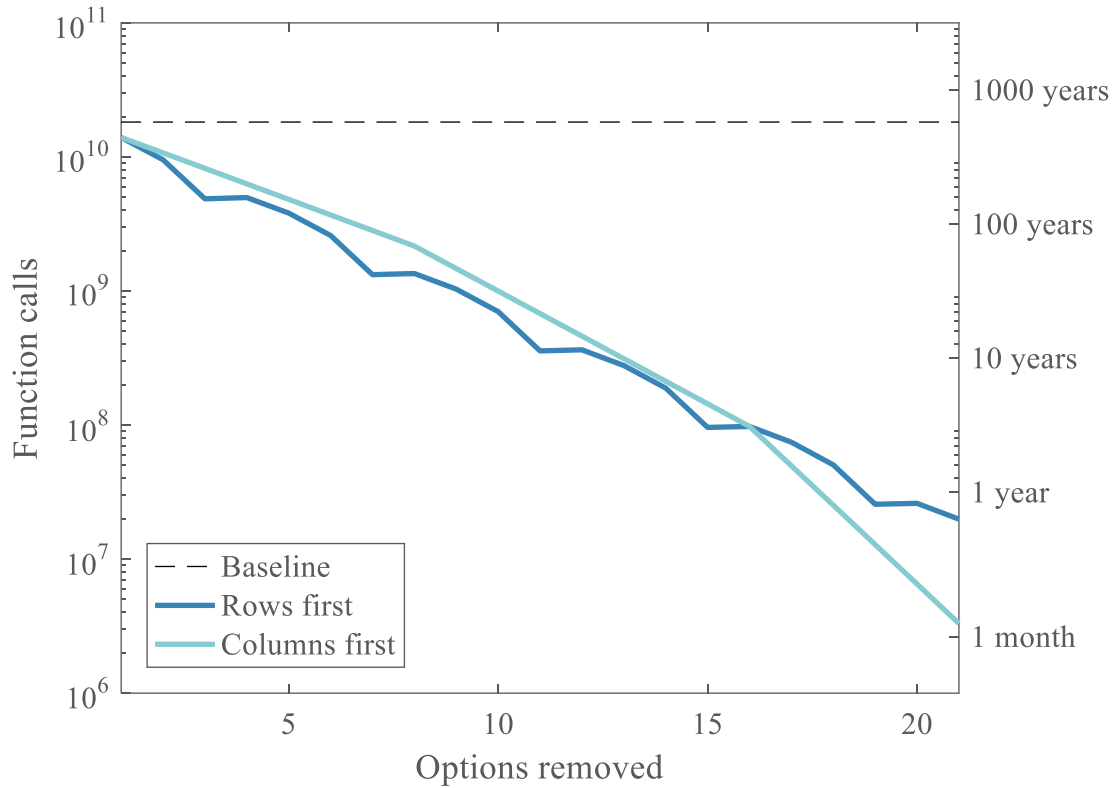


Figure 5.21: Influence of options removal strategy on morphological reduction

The “rows first” strategy seems to decrease the number of function calls the most compared to the “columns first” strategy. Jumps can be seen in both curves: for the “rows first” strategy, there is a different slope for every removed option, and the pattern repeats every time a new row is removed (every 4 options). When the last option of a row is removed, the number of function calls actually increases. Indeed, the number of discrete calls remains the same, but more variables are transferred to the optimizer. All things considered, the number of function calls hence increases slightly at every new row before decreasing again. As for the “columns first” strategy, the cycle repeats after columns are removed from the morphological matrix (every 8 options in this case).

After 16 options removed (equivalent of 2 columns, or 4 rows), combinatorial theory shows that both strategies leave the same number of alternatives. After this point, the “columns first theory” decreases the number of alternatives much faster. However, this breaking point happens when the number of options removed gets very close to the number of total options in the morphological matrix (21 options removed out of 32 total options). Finally, this graph shows that before this breaking point which happens in rare cases, both strategies exhibit a similar high-level trend: the characterization made on the previous examples hold for both strategies which generalizes the conclusions made.

Note that in real-world applications, the designer does not control any “rows first” or “columns first” options removal strategy. Instead, the options which are removed are given by the morphological reduction process and do not respect any specific pattern. The strategies were introduced for the sake of these examples so as to enable automated large-scale options removal. Moreover, the test and the notional morphological matrix were designed so that 0 to 30 percent of the total options were removed.

5.1.5.1.2 Multi-level

A complete characterization of morphological reduction for single-level design spaces was carried out in the previous subsection and it is now essential to study the repercussions when several design levels are involved. By considering a notional problem for both microscopic and macroscopic levels, the effect of removing options can be visualized for both levels. The options are removed separately on the microscopic level and the macroscopic level (Figure 5.22) and are expressed as a percentage of the size of the respective problem. In order to understand each effect separately, only the number of

variables transferred to the optimizer is considered here, the number of discrete runs is considered the same so that these graphs are equivalent to running the optimizer only once with the increased number of variables. Hence, using the surrogate model f_c defined in Equation 5.4, the formulae used are of the form $f(n_{Options\ removed}) = n_{Options}^{n_{Rows}} \times f_c(n_{vars})$ with f the number of function calls, and n_{Rows} and $n_{Options}$ corresponding to the values of the initial problem.

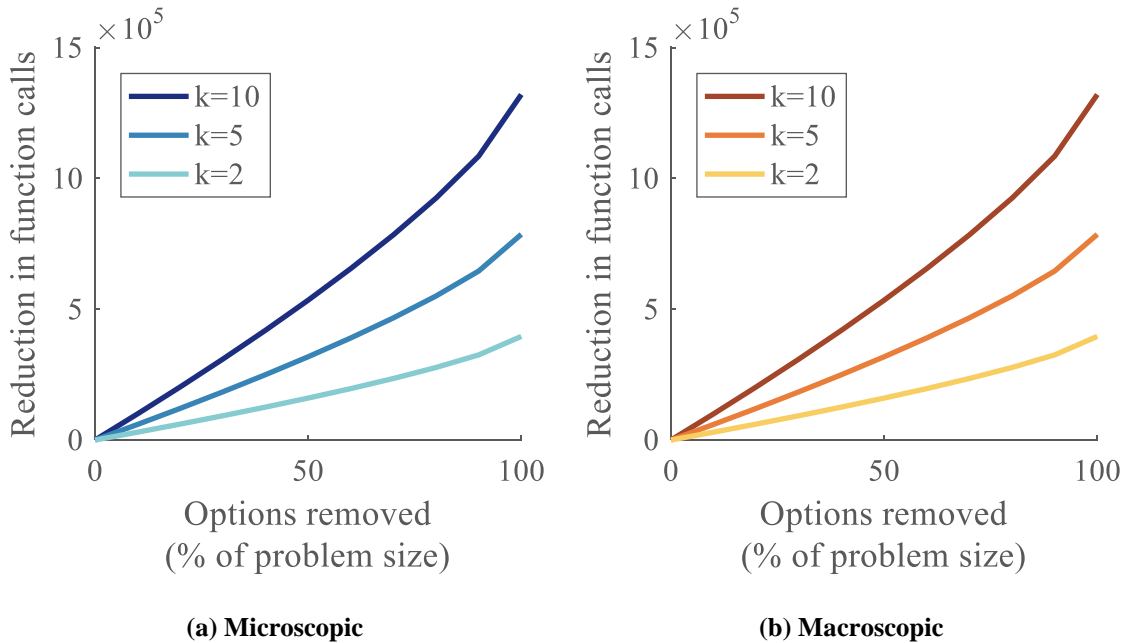


Figure 5.22: Notional morphological reduction on a bi-level problem

The decrease in the number of function calls is monotonous with the number of options removed from the given level. Since both levels are considered separately and provided that the size of the macroscopic problem is smaller than for the microscopic one, the number of function calls is decreased the most for the microscopic level. The more

options are removed, the more the impact of the k-factor is perceived: the envelope shifts away from the nominal case of $k = 5$.

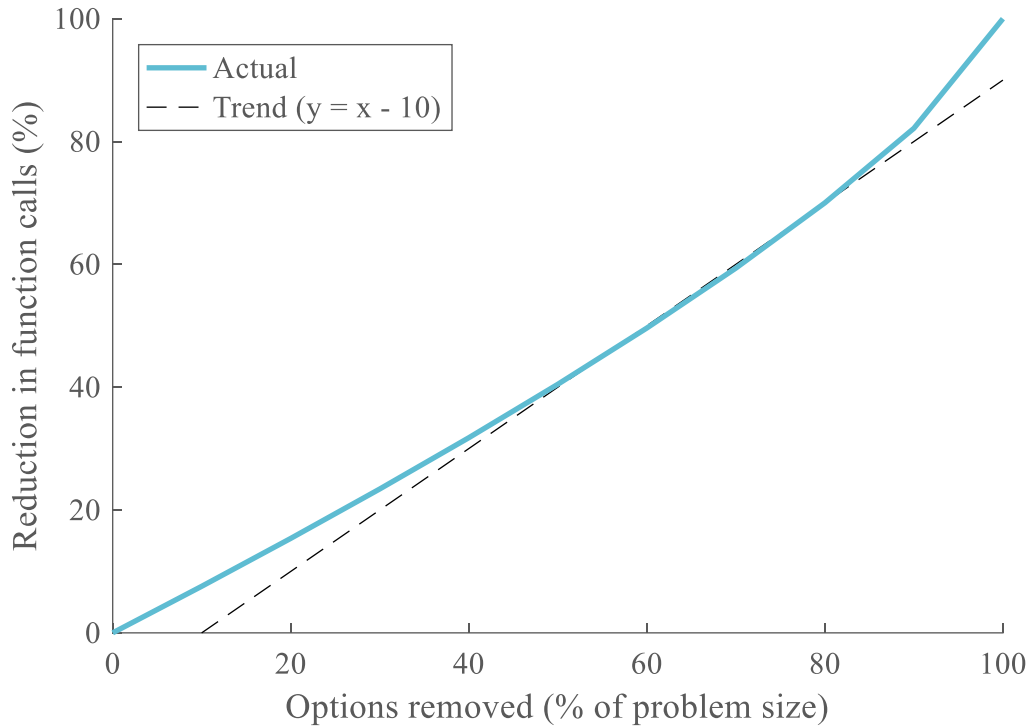


Figure 5.23: Relative morphological reduction

In terms of relative evolution, the curves are similar for any k-factor and follow the quasi-linear trend displayed in Figure 5.23. Over a large portion of the range, a change in the number of options will result in a similar change in function calls offset by 10%. This means that if 40% of options are removed in terms of the problem size, the number of function calls is reduced by around 30%.

Finally, the size of the different problems (macroscopic and microscopic) is changed on Figure 5.24.

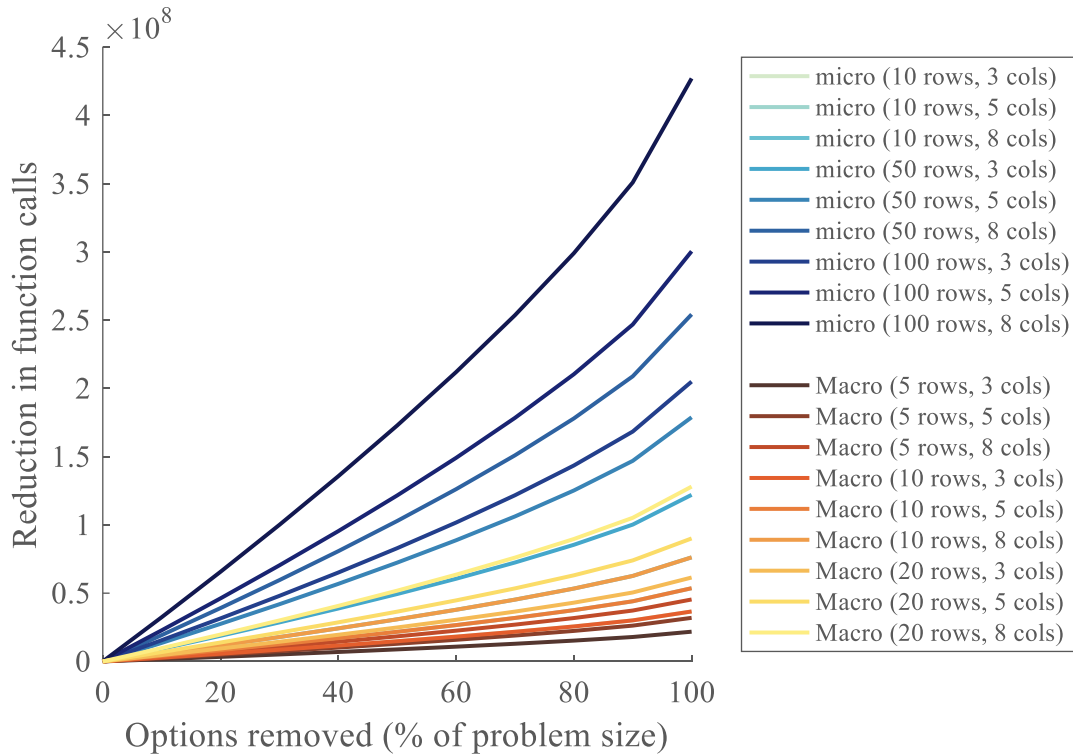


Figure 5.24: Influence of problem size on bi-level morphological reduction

As observed in real-world applications, the size of macroscopic problems (brown curves) is generally smaller than for the microscopic ones (blue curves). Hence, the number of function calls is reduced the most for microscopic problems. Some curves from the different levels overlap when they have similar morphological matrices.

Now that the effect of removing options (more precisely only adding design variables to the optimizer) has been independently characterized on both levels, it is

essential to include the effect of the reduced number of discrete calls to the optimizer to finish a complete characterization of morphological reduction. This time, to account for options being individually removed, the formulae used for each level are of the form $f(n_{Options\ removed}) = \prod_{i=1}^{n_{Rows}} n_{Options}(i) \times f_c(n_{vars})$. Note that the factor k is hidden in $n_{vars} = k \times n_{Options\ removed}$. First, the influence of this complete formula is assessed on each level with Figure 5.25. As expected, trends with cycles similar to the previous single-level section are obtained.

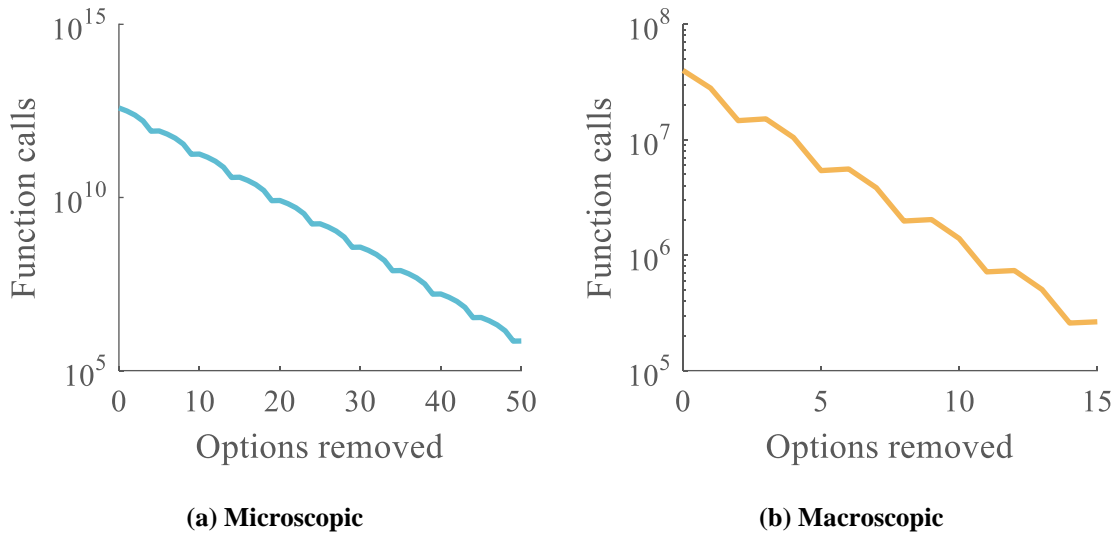


Figure 5.25: Complete morphological reduction on bi-level problem

Subsequently, the influence of both levels is merged into a unique analysis so that the effects of morphological reduction on a bi-level problem are fully understood. Including macroscopic and microscopic levels, the overall formula for the number of function calls for the whole problem is given in Equation 5.5.

Equation 5.5: General equation for bi-level morphological reduction

$$f(n_{Options\ removed}) = \underbrace{\left(\prod_{i=1}^{n_{Rows}^m} n_{Options}^m(i) \times f_c(n_{vars}^m) \right)}_{Microscopic} \times \underbrace{\left(\prod_{i=1}^{n_{Rows}^M} n_{Options}^M(i) \times f_c(n_{vars}^M) \right)}_{Macroscopic}$$

With subscripts m and M referring respectively to microscopic and macroscopic levels.

The influence of both levels is displayed on Figure 5.26.

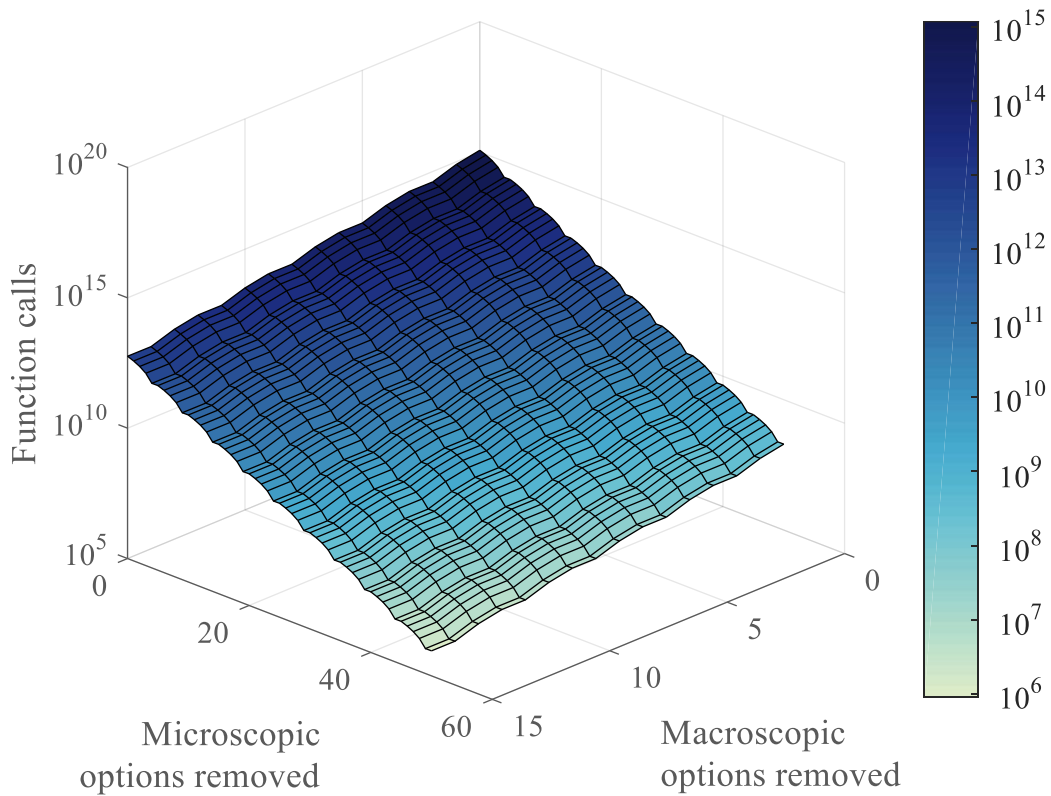


Figure 5.26: Influence of both levels on morphological reduction

The expected trend is observed, the number of function calls decreases with the number of options removed for both levels: the removal of discrete optimizer calls still dominates over the addition of design variables in the optimizer. Given the different size of both problems, the range of removed options is not the same for the two levels. In order to better capture and understand the influence of each level on this general response, a variables profiler is displayed in Figure 5.27. This profiler explores cross sections of the response across each factor around the point where one option is removed at each level. It corresponds to the partial derivatives $\left. \frac{\partial f}{\partial n_{\text{Options removed}}} \right|_{n_{\text{Options removed}}=1}$.

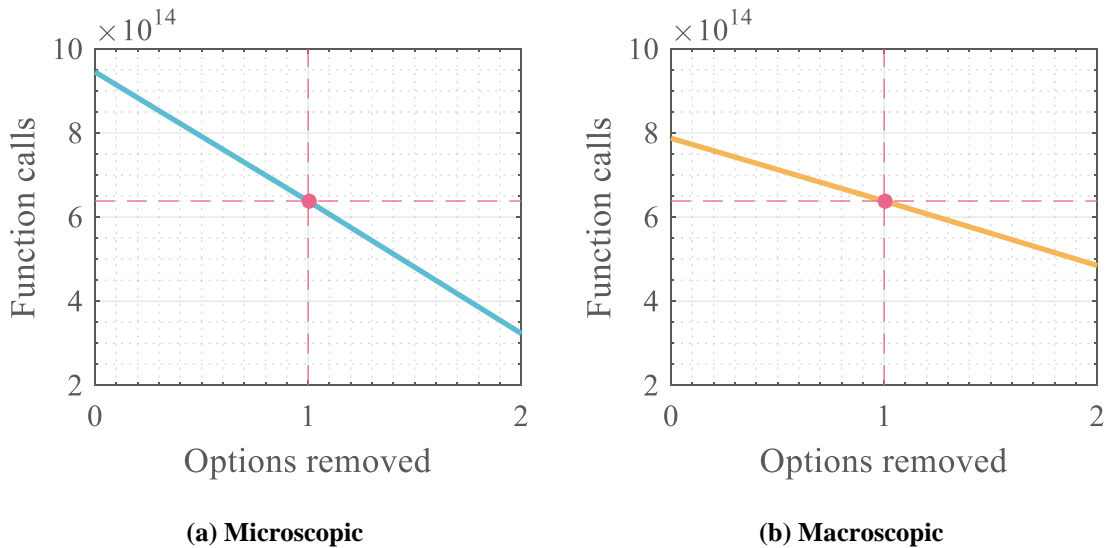


Figure 5.27: Variables profiler for morphological reduction

Using the profiler, one can now see that the macroscopic level has around twice as much influence on the global response than the microscopic level. This is consistent since the choices made at the macroscopic level have a knock-on effect on the levels below. Indeed, from Equation 5.5, the upper level multiplies the number of alternatives available

at the lower levels. Would the number of levels increase, each level would have more influence on the total number of alternatives (and total number of function calls) than the levels below it. Changing only the number of options removed from the macroscopic level, having more levels underneath is equivalent to having a bigger microscopic level (all lower levels are fixed). Hence when removing options from the upper level, increasing the number of levels at the lower levels would generate graphs similar to Figure 5.18 and Figure 5.19.

Moreover, supposing that the macroscopic level might have less design variables per option as it is often observed in real-world problems, one could envisage having different k-factors for the microscopic level and the macroscopic level. In this case the same effects as observed on Figure 5.16, Figure 5.22, and possibly even Figure 5.15 would be obtained.

5.1.5.2 Morphological tree

The morphological tree structure is trickier to quantifiably characterize than morphological reduction as it consists in a representation paradigm. Qualitatively, an example implementation of morphological tree has been proposed and its use has demonstrated clear advantages in modularity. Indeed, the morphological tree has proven easily adaptable to new architectures thanks to the morphological interfaces. Moreover, it is quite flexible in its way to handle different levels and provides an insightful graphic representation of multi-level and multi-architectures design spaces.

Thanks to the object-oriented implementation proposed on Figure 5.6, the complex process of morphological reduction at multiple levels is now reduced to traversing the

morphological tree and calling the reduction on each morphological matrix before finally computing the total number of alternatives. Finally, the tree can be used in different ways:

- As a fixed simple bookkeeping tool storing the possible design choices in an organized and graphic fashion.
- As a fully functional and dynamic design space definition tool used to lock design choices and propagate them through the tree. It can be used to incrementally perform morphological reduction, compute the total of remaining alternatives to study, as well as how many design variables were lost in the process. In particular, a possible use of the tree is to reduce a huge design space to a smaller one by generating a reduced number of architectures to study.
- As an assistant to design optimization by dynamically ensuring that the optimizers consider only feasible designs during the design space exploration.

5.2 Design optimization: the bi-level genetic algorithm

This part details the implementation and validation of the global optimization algorithm conceptually designed in section 2.3.2.2 page 163. In particular, it also performs a complete characterization of the algorithm in order to answer research question 3.2 which is recalled here with the corresponding hypothesis.

Research question 3.2

How can swarm architectures be efficiently optimized
in a multi-architecture multi-level design space?

Hypothesis 3.2

IF an optimization method based on a bi-level genetic algorithm is used
THEN a fast and efficient multi-architecture multi-level global optimization of group configurations is enabled

The associated experiment aims at demonstrating that such an algorithm is able to efficiently optimize multi-robot systems in the context of conceptual physical design.

5.2.1 Implementation

A generic representation of the bi-level optimizer is given on Figure 5.28. As opposed to Figure 2.25, note that this schematic does not assume that the optimizer is based on a genetic algorithm. This allows for further research to plug in different optimization algorithms which are more adapted to other fields. However, it was established in section 2.3.2 that genetic algorithms have desirable features for the scope of this research. Hence, an example implementation based on genetic algorithms is proposed here.

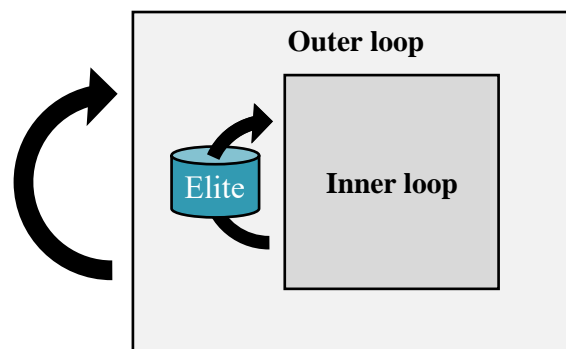


Figure 5.28: High-level architecture of the proposed bi-level optimizer

As a reminder, the outer loop decides on the swarm constitution, hence adapting the size of the design vectors required for the inner loop. The inner loop is then left to optimize remaining design variables, be it microscopic or macroscopic. The optimal microscopic architectures are kept in memory so as to be further used by future iterations. The main feature of this bi-level optimizer is that the outer loop dynamically adjusts the size of the design vectors used in the inner loop depending on the swarm constitution. Indeed, if a swarm of four vehicles is considered, the design vector of the inner loop chromosomes will have a different size than when a swarm of two is evaluated by the outer loop. This is illustrated on Figure 5.29 and formally represented on Figure 5.30.

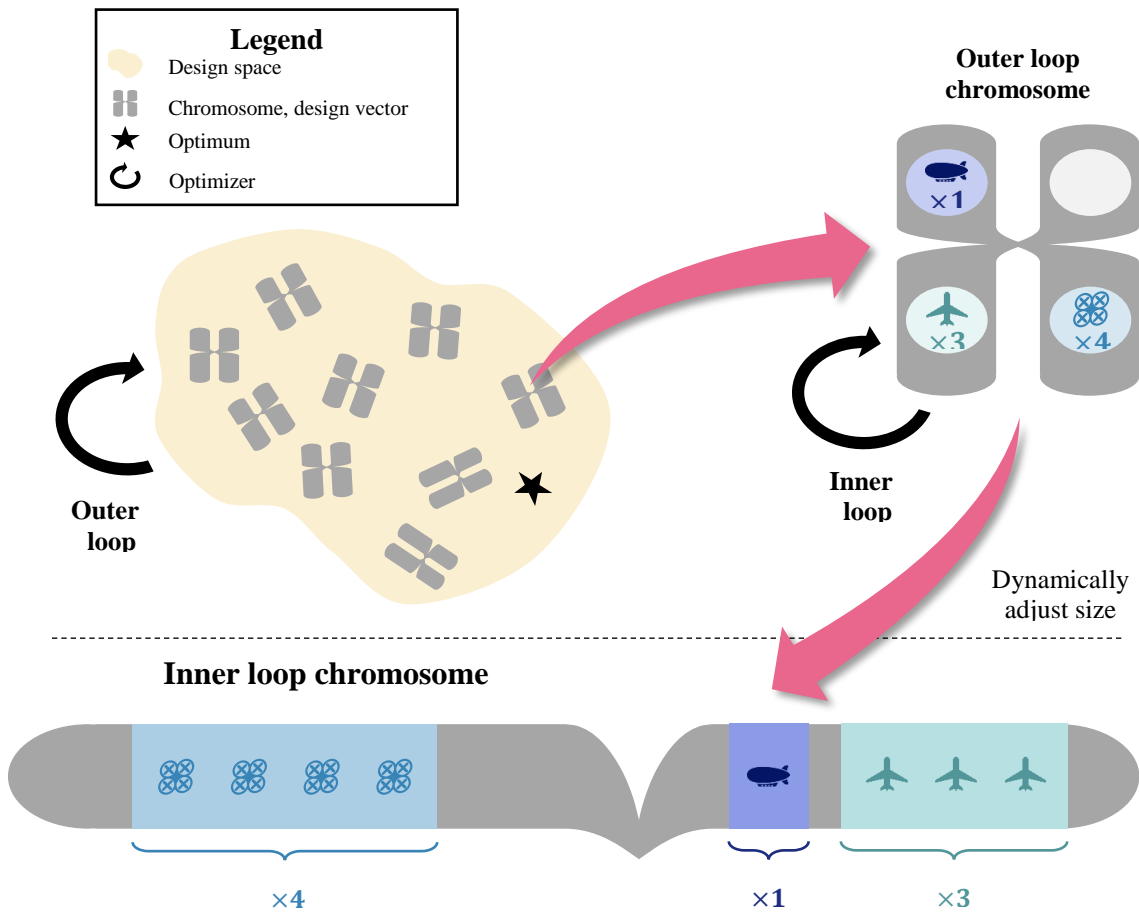


Figure 5.29: Dynamic size allocation for inner loop chromosomes

The following nomenclature is used:

- A_i Architecture i
- N Quantity of a certain object (example: N_{A_1} is the number of vehicles with architecture A_1)
- v Variables
- V Vehicle
- X Design vector (chromosome)

Outer loop

Population $\{X_{out}^1, X_{out}^2, \dots, X_{out}^{N_{out}}\}$

1 Chromosome $X_{out} = [N_{A_1}, N_{A_2}, \dots, N_{A_N}]$
 $(1, N)$

Inner loop

Fix chromosome size

Remainder of macroscopic variables

$X_{macro} = [v_{macro}^1, v_{macro}^2, \dots, v_{macro}^{N_{v_{macro}}}]$
 $(1, N_{v_{macro}})$

1 Chromosome

$X_{in} = \begin{bmatrix} X_{macro}, \\ V_{A_1}^1, V_{A_1}^2, \dots, V_{A_1}^{N_{A_1}}, \\ V_{A_2}^1, V_{A_2}^2, \dots, V_{A_2}^{N_{A_2}}, \\ \vdots \\ V_{A_N}^1, V_{A_N}^2, \dots, V_{A_N}^{N_{A_N}} \end{bmatrix}$
 $(1, N_{v_{macro}} + \sum_{i=1}^N N_{A_i} N_{v_{A_i}})$

“Vehicle i ”
 Design vector architecture i
 $V_{A_i} = [v_{A_i}^1, v_{A_i}^2, \dots, v_{A_i}^{N_{v_{A_i}}}]$
 $(1, N_{v_{A_i}})$

Population $\{X_{in}^1, X_{in}^2, \dots, X_{in}^{N_{in}}\}$

Figure 5.30: Dynamic size allocation formulae

Remembering Figure 5.28, key features are required for the optimizer:

- A memory of past optimal microscopic architectures must be initialized by the outer loop.

- This memory must be accessible and modifiable by both inner and outer loops.
- This memory must be filled with the best chromosomes (design vectors) for each architecture and hence must take into account the dynamically evolving size of the inner loop chromosomes.
- At each of its generations, the outer loop should have the possibility to alter the properties (initial population) of the instantiated inner loop algorithm (Figure 5.29).
- For a given generation of the outer loop, the chromosomes of the outer loop (i.e. the instantiated inner loops) should all be using the same initial population. This means that the initial population of the inner loops changes only after each generation of the outer loop.

Note that it is chosen that the retention of the optimal microscopic architectures is only used at each generation of the outer loop, and not for every instantiation of the inner loop. This design choice is particularly important as it ensures that the optimal microscopic designs have the time to be “averaged” over several group configurations before being ranked and used for subsequent optimizations. If this was not the case and the initial population of the inner loop chromosomes were to be updated at every instantiation of an inner loop, then the first chromosomes (inner loops) of the outer loop would be strongly biased by the constitution of the groups which were first evaluated. An explanatory example is provided on Figure 5.31.

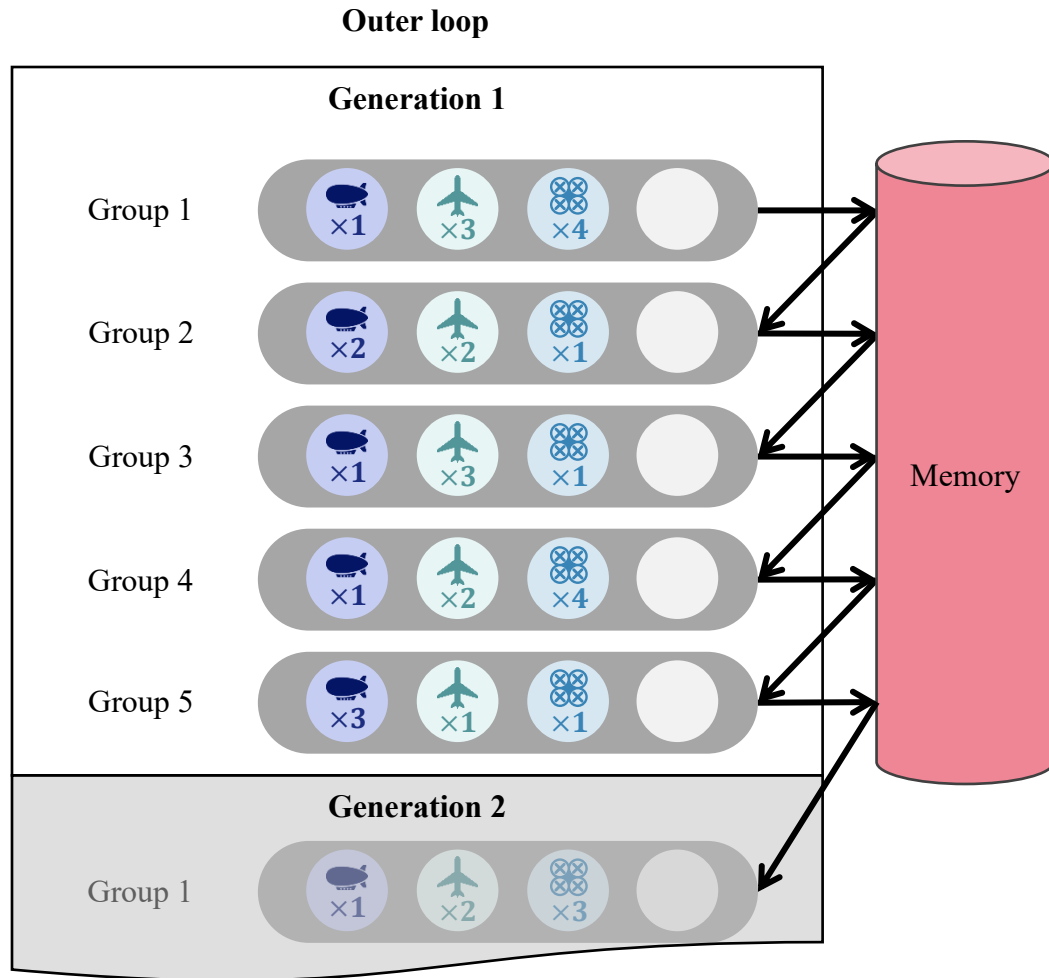


Figure 5.31: Population initialization at every instantiation

As it can be seen in this first case, if the initial population of the inner loops is updated for every instantiation of an inner loop in the outer loop population, the first chromosomes of the outer loop will be biased. Indeed, chromosome 1 exhibits a swarm composition $\{1B, 3P, 4Q\}$ (1 blimp, 3 planes, 4 quadcopters) and saves the optimal configurations in memory. When chromosome 2 is instantiated, its population is initialized with the optimal microscopic configurations from the memory. However, this memory contains only the optimal configurations for a swarm $\{1B, 3P, 4Q\}$, which might bias the

results of the optimization. As more and more chromosomes of the outer loop population are evaluated, the rankings of the optimal configurations are updated in memory and a trend emerges for the best configurations. Hence, an implementation similar to Figure 5.32 is preferred, where the optimal configurations have the time to be averaged over one complete generation of the outer loop before being used by the inner loops of the next generation.

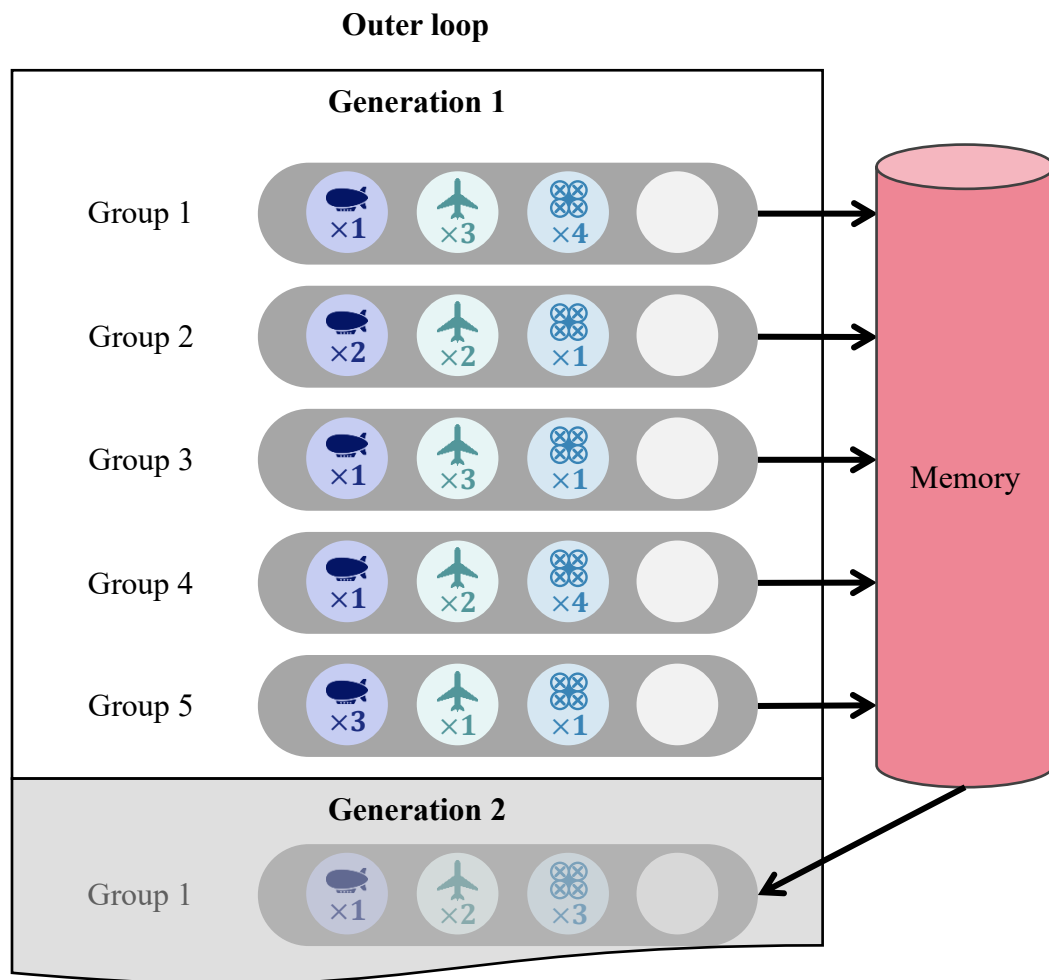


Figure 5.32: Population initialization at every generation

An implementation in Matlab is chosen for its quasi non-existent learning curve, its flexibility, and for the fact that powerful numerical, plotting, and debugging libraries are included by default in the software. In particular, the global optimization toolbox comprises a validated genetic algorithm that presents many customizable options as opposed to other possible custom or existing C++ and Java libraries which would have to be modified and validated. In addition, this genetic algorithm is able to deal with linear and nonlinear constraints.

In particular, the genetic algorithm from Matlab presents the following capabilities:

- Setting the initial population matrix, this one can be partial only.
- Setting the initial scores corresponding to the initial population. This enables to speed up the initialization of the algorithm.
- Using an output function which can be called at the end of every generation, but also at the initialization and final step of the algorithm. This can be used to save the best microscopic configurations at the end of every inner loop optimization.
- Parallelization of the execution by exploiting the different cores of the processor. This is useful to speed up the optimization process.

Based on the available options, it is possible to implement the elitism scheme using a temporary memory buffer. This buffer will accumulate and update optimal microscopic configurations after each optimization of an inner loop. Then, after a generation of the outer loop is complete, the buffer will be discharged into an “elite memory”. This elite memory remains unchanged during a generation of the outer loop. As a consequence, during one

generation of the outer loop, all inner loops can be instantiated from this elite memory as they would all use the same elite as their initial population (Figure 5.33).

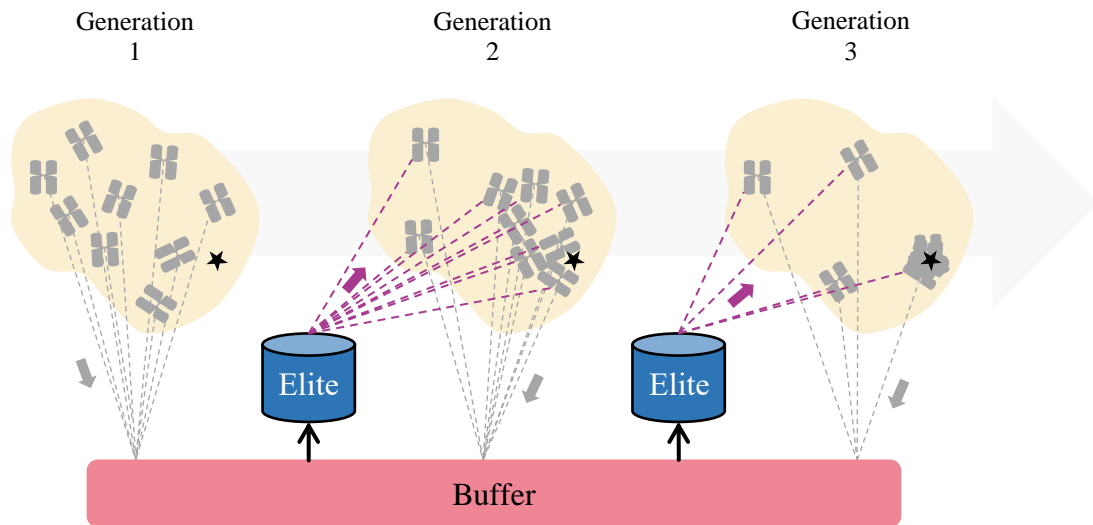


Figure 5.33: Elite retention with elite memory

Note that if only a memory buffer is used, then there is no guarantee that the inner loops are initialized with the same elite during one generation of the outer loop (see Figure 5.34).

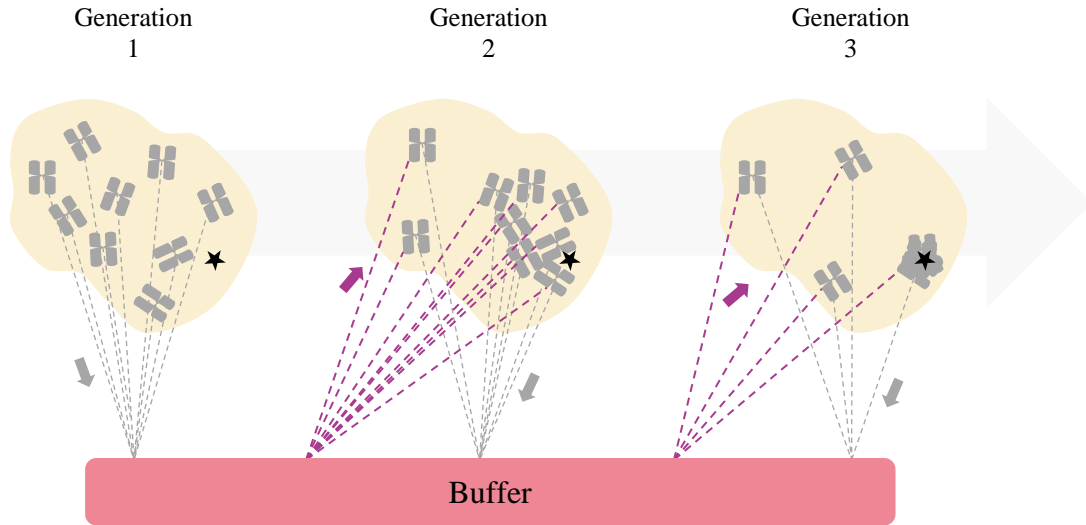


Figure 5.34: Elite retention with buffer only

Constraints: in addition to the constraints placed by the designer on the swarm optimizer (swarm cost, time of the mission, etc.) a few constraints have to be imposed on the optimizer. The first concern is to speed up the algorithm by detailing the domain it is working in. Hence, the designer has to input a maximum number of vehicles which is used for both inequality and upper bound constraints (see Equation 5.6). On top of that, one constraint also insures that there is at least one vehicle in the swarm.

Equation 5.6: Overall numerality constraints

$$1 \leq \sum_{i=1}^{N_{arch}} X_i^{out} \leq N_{max}$$

Finally, additional constraints ensure that the number of vehicles for each architecture remains within relaxed bounds and that it is an integer number. Indeed, since

the total number of vehicles in the swarm cannot exceed N_{\max} , each architecture must also respect this inequality (Equation 5.7).

Equation 5.7: Individual numerality constraints

$$\begin{cases} 0 \leq X_i^{out} \leq N_{\max}, \forall i \in \llbracket 1, N_{arch} \rrbracket \\ X_i^{out} \in \mathbb{N} \end{cases}$$

Hence, the optimization problem of the outer loop can be formally written as:

Equation 5.8: Outer loop optimization problem

$$\begin{aligned} & \min_{X^{out}} in(X^{out}) \\ & \text{subject to} \\ & \left\{ \begin{array}{l} 1 \leq \sum_{i=1}^{N_{arch}} X_i^{out} \leq N_{\max} \\ 0 \leq X_i^{out} \leq N_{\max}, \forall i \in \llbracket 1, N_{arch} \rrbracket \\ X_i^{out} \in \mathbb{N}, \forall i \in \llbracket 1, N_{arch} \rrbracket \end{array} \right. \end{aligned}$$

Where in is the inner loop, a complete other optimization algorithm of its own. This inner loop optimization problem can be written as:

Equation 5.9: Inner loop optimization problem

$$\begin{aligned} & \min_{X^{in}} f(X^{in}) \\ & \text{subject to} \end{aligned}$$

$$\begin{cases} c(X^{in}) \leq 0 \\ l_b \leq X^{in} \leq u_b \end{cases}$$

Where f and c are respectively a fitness and cost function for the given swarm. In particular, one can refer to section 3.1.3 page 195 to build these functions.

Special case of full heterogeneity: this research makes a distinction between what is referred to as “full heterogeneity” and “partial heterogeneity”. For the scope of this research, partial heterogeneity is defined as the case where all vehicles of a given architecture have the same configuration. Besides, with full heterogeneity, each vehicle may have its own configuration. As an example, a swarm of quadcopters will be partially heterogeneous if the quadcopters of the group are all the same. This swarm will be fully heterogeneous if each quadcopter is different (see Figure 5.35).

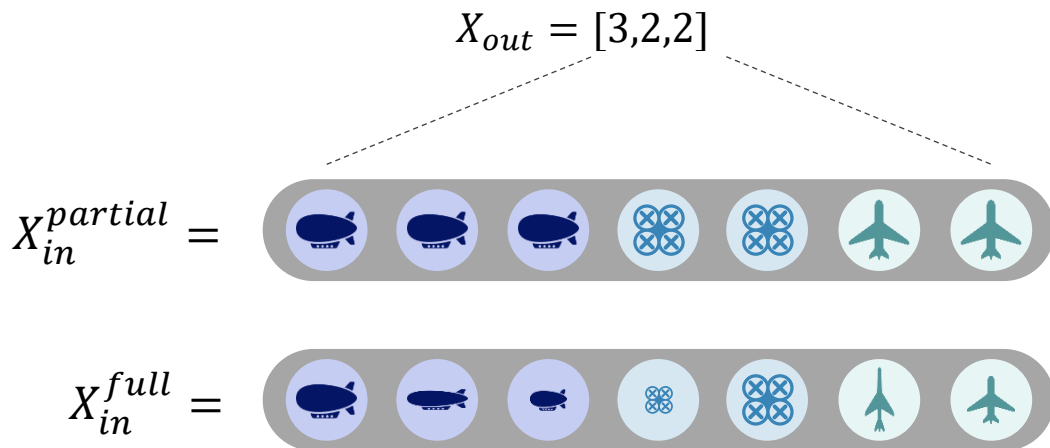


Figure 5.35: Full vs. partial heterogeneity

In the previous figure, the first swarm has vehicles of a given architecture with the same configurations, whereas the second swarm has different vehicles for each architecture.

Observing this particularity, the optimization can be greatly sped up when the designer wants to consider only partial heterogeneity. Indeed, the size of the inner loop design vector can be fixed in that case, hence also removing the need for an outer loop. An option is included in the algorithm to enforce full or partial heterogeneity and to reduce the size of the inner loop design vector when required. One may note however that the outer loop cannot be completely removed since special cases might require to further adapt the size of the inner loop design vector. This is the case when an architecture does not have any representative vehicle in the swarm.

Special case of 0: an additional situation where the optimization can be sped up is when the outer loop optimizer proposes a swarm with zero vehicles of a given architecture. In this case, the size of the inner loop design vector can be reduced by deleting all the design variables of the corresponding architecture. This is already taken into account in the case of full heterogeneity (see formulae in Figure 5.30) but has to be implemented in the case of partial heterogeneity when the number of variables is fixed. By doing so, the genetic algorithm of the inner loop will act only on a limited number of design variables, hence accelerating the optimization process.

Note that in the case where the size of the inner loop vector is dynamically arranged to accommodate for either true heterogeneity or architectures with no representative vehicle, the algorithm also has to adapt the constraint vectors as well.

Outer loop speedup: the aforementioned reduction in size of the inner loop design vectors has a direct impact on the rapidity of the inner loop, and hence indirectly helps in speeding up the convergence of the outer loop. Nonetheless, it is possible to directly improve the performance of the outer loop in specific cases. This approach considers the total number of function calls required by the genetic algorithm and observes that in some cases, using a full factorial approach for the outer loop is way more beneficial. Indeed, the population of the genetic algorithm might comprise many more individuals (and hence function calls) than if each possible swarm configuration was evaluated. We consider here the example of an outer loop with a population of 50 individuals, if the outer loop converges in 50 generations for instance, the total convergence of the algorithm requires 2500 function calls. However, if the swarm to be configured has only 3 architectures, and 10 agents at maximum in the group, the optimization problem given in Equation 5.8 yields $11^3 = 1331$ possible cases for the full factorial approach. Indeed, each element of X^{out} can take values between 0 and 10. By removing the cases with more than 10 total agents in the group, the full factorial approach is reduced to 286 cases or evaluations of the inner loop: a number lower than the case of the genetic algorithm by two orders of magnitude.

It is then possible to study this trend for an evolving number of architectures and for the specific data mentioned hereinabove and 12 agents at maximum in the group, the trends shown in Figure 5.36 are obtained.

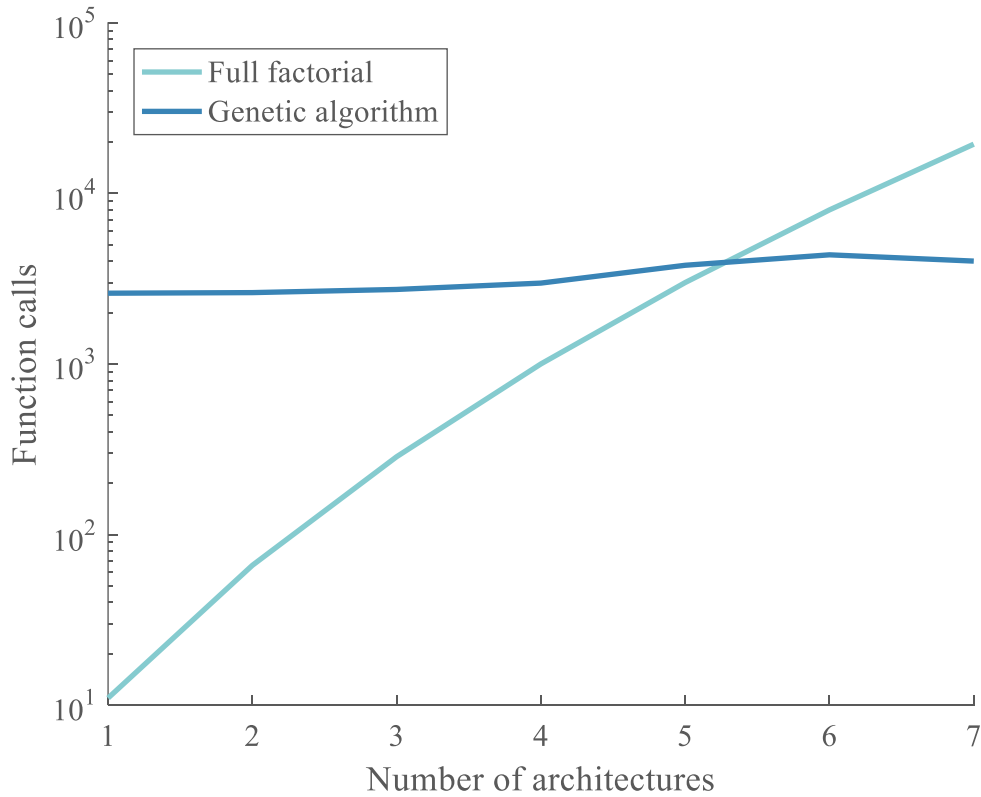


Figure 5.36: Genetic algorithm vs. full factorial

In that particular case, it can be seen that a full factorial approach would perform much faster in finding the optimum of the outer loop for swarms comprising less than 5 architectures. It can be noticed that the number of function calls for the genetic algorithm does not vary much with the number of architectures considered. Indeed, the stopping criteria of the genetic algorithm is based on the number of stalled generations: the number of generation for which the best individual does not change. Hence if this criterion is fixed

at 50 generations, the algorithm will generally take a few generations to find the best individual and then will stall. There is no surprise then that, in this case, the total number of generations would be always slightly above 50 (with a corresponding number of function calls). These remarks are leveraged in the implementation of the algorithm since the choice is given to the designer to opt for a full factorial or a genetic algorithm optimizer for the outer loop. In order to make this choice, the designer should compute the total number of alternatives generated by a full factorial approach based on the maximum number of vehicles in the swarm, and then delete the rows of the full factorial matrix which do not respect this constraint. However, due to the important memory required to compute full factorial design matrices for more than 10 factors (and 10 levels for instance), this computation might not be always possible. Consequently, a crude approach can be used where the design does not eliminate the rows for which the sum does not satisfy the constraint imposing the maximum number of robots in the swarm. The designer does not have to generate the matrix anymore and can simply compute the number of alternatives using the N_{\max}^{Arch} formula. Using this simplified formula (Figure 5.37), he can decide whether the genetic algorithm or the full factorial approach will be faster by assuming or enforcing a maximum number of generations for the outer loop.

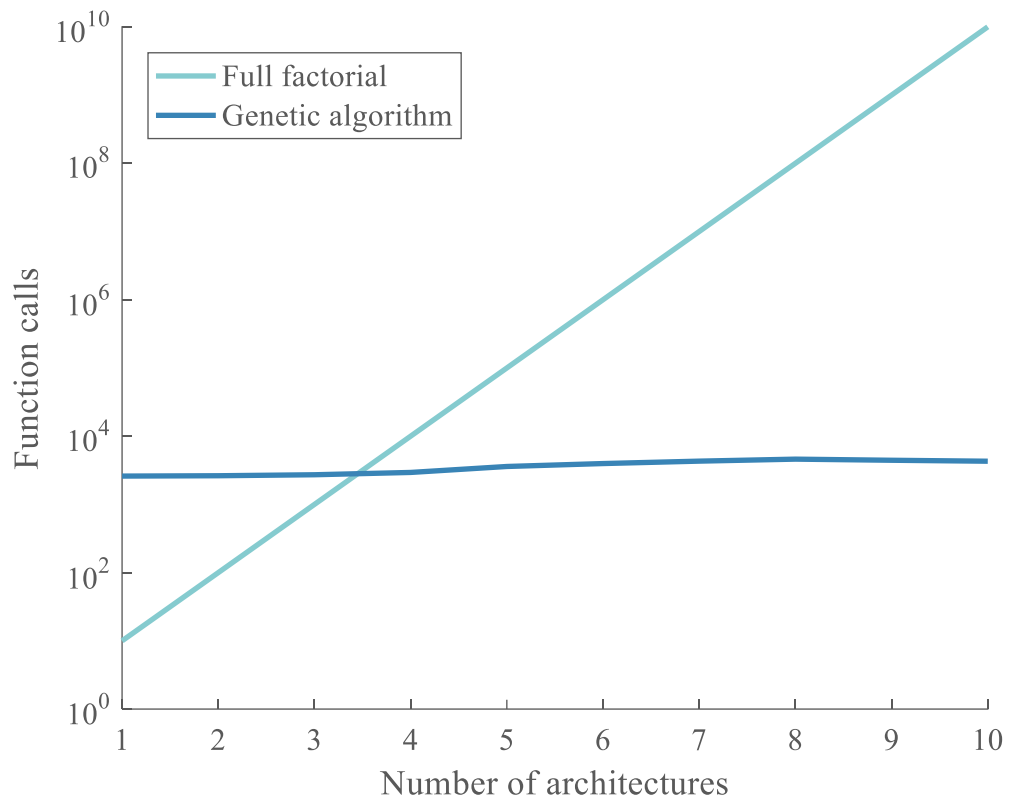


Figure 5.37: Simplified expression for full factorial approach

Using this simplified approach, the design finds that it is more valuable in terms of computation time to use the full factorial approach for swarms under 4 architectures, compared with 5 obtained with the exact computation. Note this time the linear evolution in logarithmic space of the full factorial approach ($N_{\max}^{N_{arch}}$).

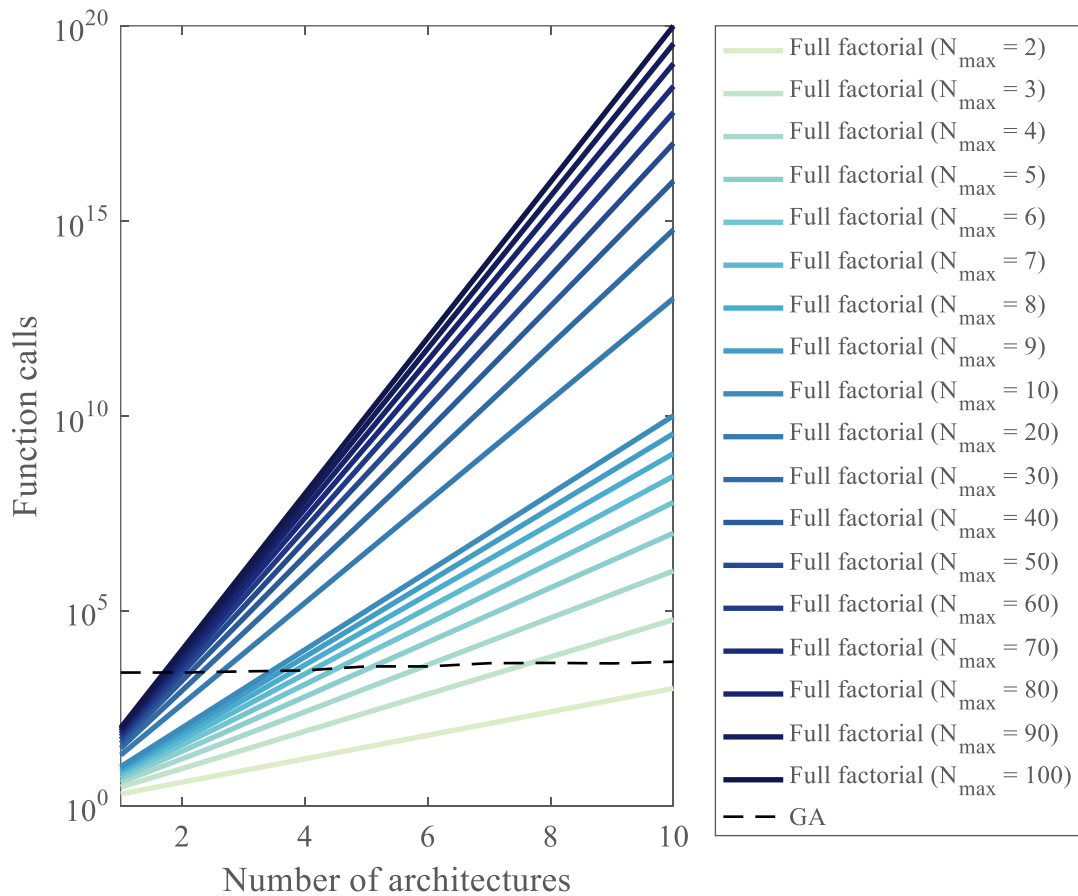


Figure 5.38: Choosing between full factorial or genetic algorithm

By varying the maximum number of agents in the swarm and using the simplified expression discussed earlier, it is possible to obtain the map displayed in Figure 5.38. The function calls for the genetic algorithm have been averaged over 10 replications. Depending on the number of architectures considered (up to 10), and the maximum number of agents in the swarm, the designer is able to see which approach from the full factorial or the genetic algorithm is more efficient for the outer loop. In particular, several observations can be made:

- If only one architecture is to be considered, the full factorial approach is always more profitable in terms of computation time.
- Most of the variation is observed for N_{\max} values below 7. After that, it takes considerable changes in N_{\max} to observe an evolution in the intersection with the curve of the genetic algorithm.

Algorithm input: the inputs and main parameters of the bi-level swarm optimizer are detailed here below:

- Number of architectures
- Number of variables for each architecture
- Maximum number of agents for the swarm
- Outer loop solver: genetic algorithm or full factorial (see previous remarks)
- Type of heterogeneity: full or partial
- Population sizes for both outer and inner loop
- Use of elite retention and associated elite fraction
- Inner loop fitness function
- Inner loop constraint function
- Additional options: mostly options embedded in the Matlab genetic algorithm
 - Possible plot and output functions for each iteration of the algorithm or once the algorithm has finished. For instance, these can be functions that save iteration information into a file for later use.
 - Stopping criteria: based on the number of stalled generations
 - Use of parallelization or not

- Function and constraint tolerances

Example use of the algorithm and example values for these parameters are given in the code appendix (see Appendix B.2 page 493).

5.2.2 Verification and validation

Before using the bi-level genetic algorithm, it has to be verified and validated using test functions with a known behavior. This step will ensure that the method is implemented as per the requirements and that it works properly when it is used for the optimization of groups of robots.

5.2.2.1 Test function

The optimization scheme has to be tested against a function which response and behavior is known. This will enable the verification of the algorithm to see if it is able to predict the correct optimum. Nonetheless, no test function seems to exist for the optimization of dynamic design spaces. Hence, an analytical test function is proposed here to simulate the performance of a group of robots depending on its possible variables.

Before starting to build the test function, a few observations are required:

- A swarm is composed of architectures
- Each architecture in the swarm is represented by one or several vehicles
- Each vehicle is composed of design variables
- Each of these vehicles can be unique and have its own design variables

It is essential for the test function to provide different optima for different group configurations so that each case can be verified independently and with certainty. Consequently, the test function has to take into account all the different variables and factors mentioned hereinabove. The optimum (value and location) of the test function should then depend on:

- The number of architectures considered
- The number of vehicles in the group for each architecture
- The number of design variables per vehicle
- Optional: The number of extra macroscopic design variables

Outer loop: the outer loop has to optimize the number of vehicles for each architecture. An easy test function for the outer loop is then to have an optimum that corresponds to the index of each architecture. Hence, if three architectures are considered with respective indices 1, 2, and 3; the optimum will be to have 1 vehicle of architecture 1, 2 vehicles of architecture 2 and 3 vehicles of architecture 3. This is achievable with the following example test function:

Equation 5.10: Outer loop test function

$$f_{out}(x) = - \sum_{i=1}^{N_{arch}} \frac{1}{1 + (x_i - i)^2}$$

With N_{arch} the number of architectures and x the outer loop design vector representing the composition of the swarm. For instance, $x = [2 \ 1 \ 3]$ represents a group

with 2 vehicles of architecture A_1 , 1 vehicle of architecture A_2 , and 3 vehicles of architecture A_3 . Each architecture i is given an offset of $\frac{-1}{1+(x_i-i)^2}$ which is minimal when $x_i = i$. Hence the function f_{out} will be minimized when the design vector corresponds exactly to the indices of the architectures (see Table 5.13 and Figure 5.39). This ensures that the optimum is unique when given a certain number of architectures and this can be used to verify the implementation of the outer loop.

Table 5.13: Outer loop test function values

X_{out}	Offset 1	Offset 2	Offset 3	Total $f_{out}(X_{out})$
	$\frac{-1}{1+(x_1-1)^2}$	$\frac{-1}{1+(x_2-2)^2}$	$\frac{-1}{1+(x_3-3)^2}$	
[1, 1, 1]	-1.0	-0.5	-0.2	-1.7
[1, 2, 1]	-1.0	-1.0	-0.2	-2.2
[3, 1, 3]	-0.2	-0.5	-1.0	-1.7
[1, 2, 3]	-1.0	-1.0	-1.0	-3.0
[2, 1, 3]	-0.5	-0.5	-1.0	-2.0
[3, 2, 1]	-0.2	-1.0	-0.2	-1.4

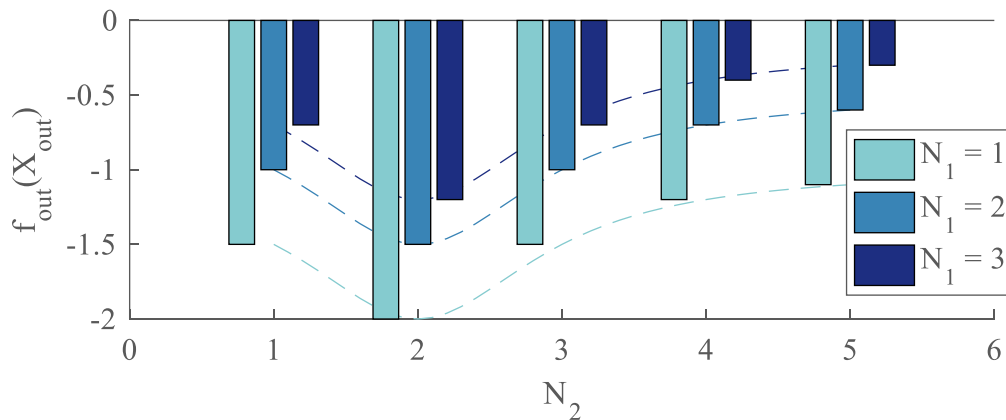


Figure 5.39: Outer loop test function offsets

For a swarm composed of only two architectures, this can also be represented graphically by plotting the value of the offset $f_{out}(X_{out})$ for different values of N_1 and N_2 , respectively number of vehicles for architecture 1 and 2. The offset is minimized when $N_1 = 1$ and $N_2 = 2$ (see Figure 5.39) and the dashed lines represent the continuous versions of the different offsets.

Inner loop: the inner loop has to take into account additional parameters in order to make the value of the optimum unique. Indeed, for a given variable of the inner loop design vector, its optimal value should depend on the architecture it belongs to, the vehicle it belongs to, and the number of variables in the vehicle. Moreover, for the location of the optimum to be unique for each swarm configuration, the optimum value of this design variable should also depend on the number of agents in the swarm. This will also insure that the microscopic and macroscopic variables are intertwined in the analysis just as they would be in the performance of a robotic swarm. One easy implementation is to have each one of this factors represented by one decimal of the optimal design variable (Equation 5.12) and use a parabola centered around this design variable (Equation 5.11). Using a simple parabola around the optimum of each design variable makes it easy to have a unique optimal value for each variable. In addition, the complexity is limited since the cost function to be minimized is polynomial of second order.

Equation 5.11: Inner loop test function

$$f_{in}(x) = \sum_{i=1}^{N_{vars}} (x_i - x_i^*)^2$$

With N_{vars} the number of variables in the design vector x , x_i the i -th component of x , and x_i^* the optimal value for a given design variable x_i . The expression for x_i^* is:

Equation 5.12: Optimal value for design variable i

$$x_i^* = A_i + \frac{N_{A_i}}{10} + \frac{V_i}{100} + \frac{i}{1000}$$

With A_i the index of the architecture corresponding to design variable i , V_i the index of the corresponding vehicle, and N_{A_i} the total number of vehicles with architecture i constituting the swarm. Before giving an example, the nomenclature for the design variables has to be presented with the ordering used in the design vectors. Hence the nomenclature for the design variables is defined as follow:

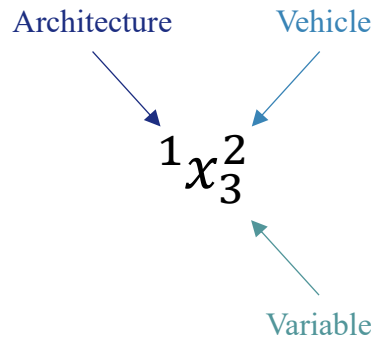


Figure 5.40: Design variables nomenclature

From Figure 5.40, that design variable is associated to the second vehicle in the swarm with architecture 1 and it is the third design variable of this vehicle. Moreover, the following ordering is used in the design vectors of the inner loop:

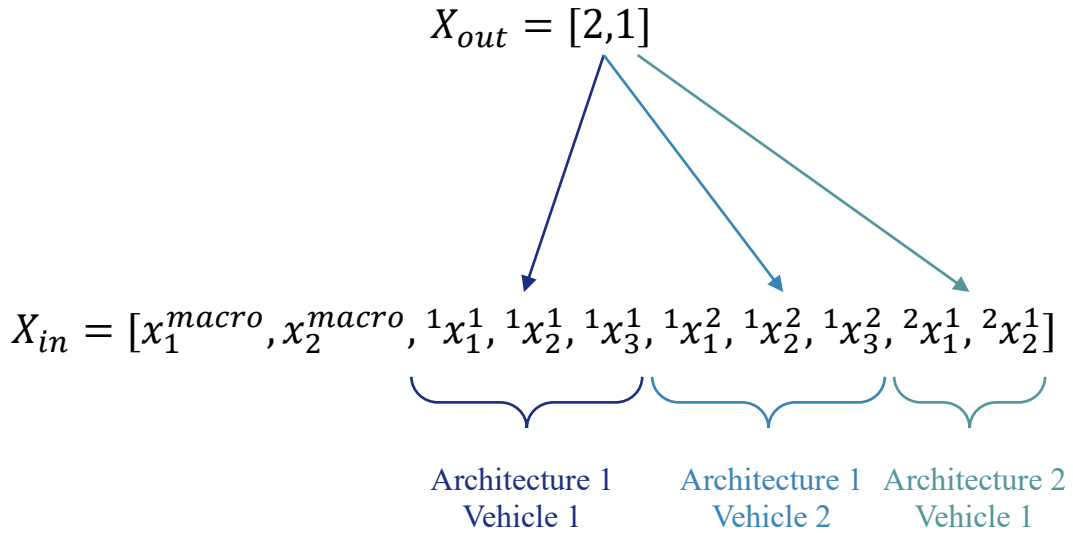


Figure 5.41: Variables ordering in the inner loop design vectors

It can be seen that, as established in Figure 5.29 and Figure 5.30, the size of the inner loop design vector is fixed by the outer loop design vector. Knowing that we have 2 macroscopic variables, 3 variables for the first architecture and 2 for the second one, the total number of variables for the inner loop chromosome can be computed. In the light of this new nomenclature, the optimal value for a given design variable $i x_k^j$ can be re-written as shown in Equation 5.13:

Equation 5.13: Optimal value for the test function variables

$$i x_k^j = i + \frac{N_{tot}}{10} + \frac{j}{100} + \frac{k}{1000}$$

This function assumes, for verification purposes, that $N_{tot} < 10$, $j < 100$, and $k < 1000$.

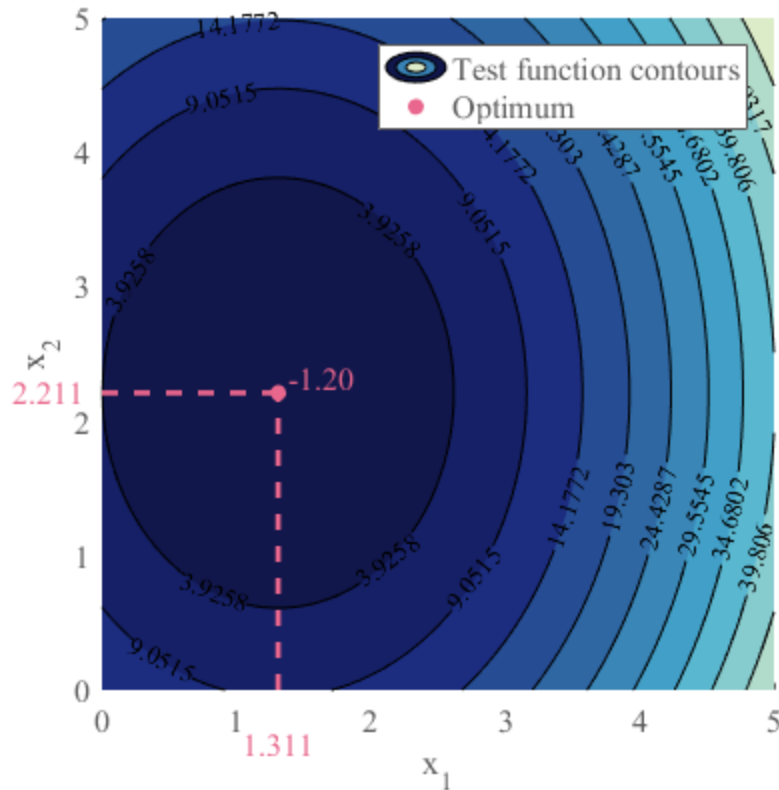


Figure 5.42: Inner loop test function visualization

Figure 5.42 shows the inner loop plot function for a given swarm composition corresponding to $X_{out} = [3,2]$. For display purposes, each architecture has only one design variable and heterogeneity is partial so that all vehicles from a given architecture have the same configuration. Hence, all vehicles of architecture 1 can be represented by design variables x_1 and the same goes for architecture 2 and x_2 . Based on these assumptions and the formula for $i x_k^j$, the inner loop test function will be minimized for $x_1 = 1.311$ (architecture 1, 3 total vehicles, vehicle 1, design variable 1) and $x_2 = 2.211$ (architecture 2, 2 total vehicles, vehicle 1, design variable 1). The evolution of the function is parabolic

around this optimum as it can be observed on Figure 5.43 by taking a section of the contour plot.

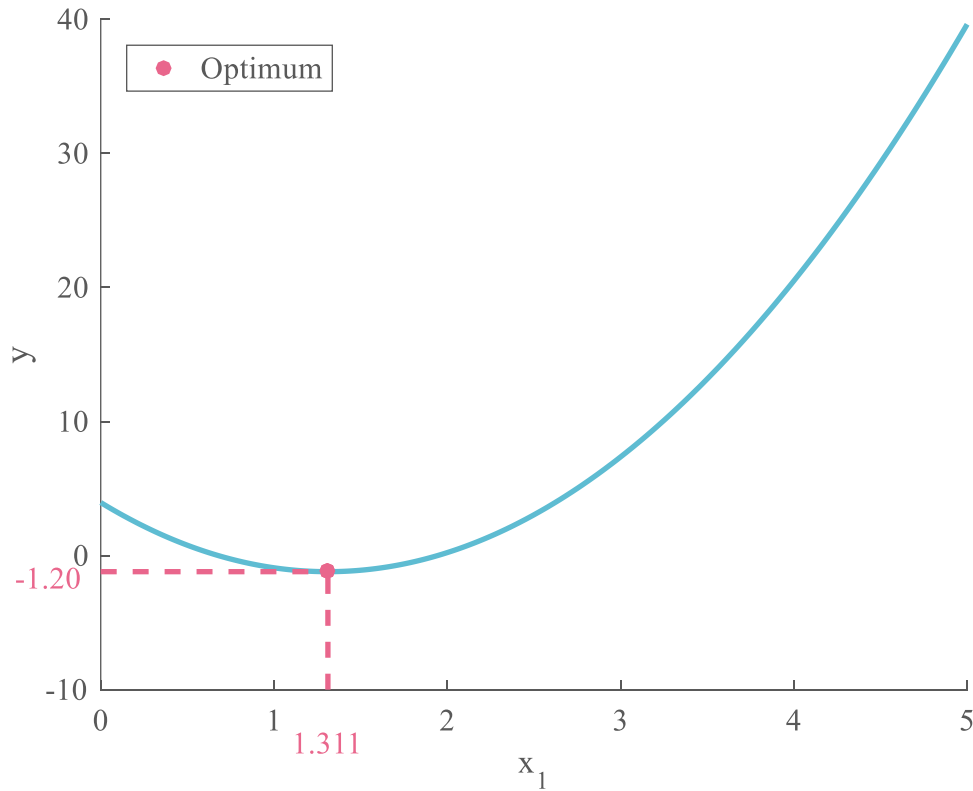


Figure 5.43: Inner loop test function sectional cut

Complete algorithm: to verify the behavior of both loops working together, it is possible to simply combine the test functions for the outer loop and the inner loop. Indeed, by adding the outer loop function to the inner loop one, we obtain a function where the inner loop variables and outer loop variables are separated. The inner loop optimizer will first try to reach the optimal value of the f_{in} part, while the outer loop will act on f_{out} to minimize the overall function. Note that a small part of f_{in} still accounts for the total number of

agents in the swarm, which is a macroscopic or outer loop variable. This part was included in f_{in} to guarantee the unicity of the optimum when verifying the inner loop and does not interfere when put together with f_{out} .

Equation 5.14: Test function for the complete algorithm

$$f(X = [x_{in}, x_{out}]) = f_{in}(x_{in}) + f_{out}(x_{out})$$

$$= \left(\sum_{i=1}^{N_{vars}(x_{in})} (x_{in}(i) - x_i^*)^2 \right) - \left(\sum_{i=1}^{N_{arch}} \frac{1}{1 + (x_{out}(i) - i)^2} \right)$$

Note that additional cross terms can possibly be added to this validation function to reinforce the correlations between microscopic and macroscopic variables and further step away from possible sequential optimization caveats. However, this would make it slightly more difficult to analytically predict the optimal value and location. A complete derivation of the global optimum of this verification function is provided in Appendix A.1 page 416.

5.2.2.2 Algorithm verification

Using the different test functions defined in the previous subsection, the implementation of the bi-level genetic algorithm can be verified using a unit test framework. A complete test suite is coded by calling the optimizer on given configurations and comparing the obtained optimal values with the ones expected thanks to the analytical approach.

Outer loop: the outer loop can be tested by using f_{out} as the fitness function. The test generates a random number of architectures and lets the outer loop derive an optimal swarm

composition. The optimum obtained from this test is then compared with the theoretical value given by the test function. For instance, if the test chooses to use 7 architectures, then the expected optimal configuration is $X_{out}^* = [1,2,3,4,5,6,7]$. Both solvers are tested: the genetic algorithm approach as well as the full factorial method. In addition, two tests are designed: one for partial heterogeneity and one for full heterogeneity. The tests were repeated one thousand times each and all tests passed.

Inner loop: the unit test for the inner loop uses the complete function f for the validation of the inner loop. A random number of architectures is generated and based on that, a random design vector X_{out} is created as well as a random number of design variables for each one of the architectures. This is enough information to determine the expected optimal values for each one of the design variables and compare them to what is obtained by running the inner loop optimizer. Both full and partial heterogeneity are tested and the inner loop is validated after a thousand successful runs of each one of the tests.

Complete algorithm: to perform a unit test for the overall algorithm, the outer loop now uses the inner loop as its fitness function so that both loops are now interconnected. The fitness function of the inner loop is the same validation function f used for the verification of the inner loop. The setup for this unit test is quite similar to the verification of the outer loop: a random number of architectures is generated as well as random numbers of variables for each of these architectures. The swarm configuration is optimized to obtain optimal vectors x_{in}^* and x_{out}^* which are compared to the theoretical values described in the previous section. This verification also tests for the different parameters of the whole

optimizer: the type of outer loop solver (genetic algorithm or full factorial), the type of heterogeneity (full or partial), and the elitism fraction. Each one of these tests passes, thus validating the behavior of the complete bi-level optimization algorithm.

Actively constrained optimization: note that so far, the algorithm has been tested with non-binding constraints. Indeed, at convergence, the optimal values of X^{out} and X^{in} are within the ranges defined in section 5.2.1. However, it is highly probable that in real-world applications, the optimization will be subject to contradictory cost and performance constraints which will most likely be active at the optimum. In order to prove the versatility of the bi-level optimizer with active constraints, an additional test can be carried out. This test involves introducing constraints on the design variables of the verification function. In order to continue the verification of the algorithm, these constraints should be designed so that they are active at the optimum, and that the constrained optimum is still relatively easy to derive analytically depending on the parameters of the problem. This will ensure that the values obtained by theory on one hand, and by the algorithm on the other hand, will be comparable and identical.

A possible solution is to offset the optimal values X_i^{*out} and X_i^{*in} , which are known values, by a constant number. The new constrained optimal values are noted as X_i^{cout} and X_i^{cin} . This new optimum is then compared with the output of the bi-level optimizer for the new optimization problem presented in Equation 5.15 and Equation 5.16.

Equation 5.15: Constrained outer loop optimization problem

$$\begin{aligned} & \min_{X^{out}} in(X^{out}) \\ & \text{subject to} \\ & \left\{ \begin{array}{l} X_i^{out} \geq X_i^{*out} + 1, \forall i \in \llbracket 1, N_{arch} \rrbracket \\ 1 \leq \sum_{i=1}^{N_{arch}} X_i^{out} \leq N_{max} \\ 0 \leq X_i^{out} \leq N_{max}, \forall i \in \llbracket 1, N_{arch} \rrbracket \\ X_i^{out} \in \mathbb{N}, \forall i \in \llbracket 1, N_{arch} \rrbracket \end{array} \right. \end{aligned}$$

Equation 5.16: Constrained inner loop optimization problem

$$\begin{aligned} & \min_{X^{in}} f(X^{in}) \\ & \text{subject to} \\ & \left\{ \begin{array}{l} X_i^{in} \geq X_i^{*in} + 1, \forall i \in \llbracket 1, N_{vars}(X^{in}) \rrbracket \\ l_b \leq X_{in} \leq u_b \end{array} \right. \end{aligned}$$

With $N_{vars}(X^{in})$ the number of variables for the inner loop design vector. In this particular example, a unit offset of one is chosen and the constrained optimum is simply $X_i^{c^{out}} = X_i^{*out} + 1$. A complete proof is given in Appendix A.1.2 (see page 422).

In the same fashion as the previous unit tests, this one generates a random number of architectures, a random number of variables for each of these architectures, and a random offset from the unconstrained optimum. This unit test also passes, hence finishing the complete verification of the optimization algorithm, including with active constraints. One can note that this particular test could have possibly been skipped given that the genetic algorithm from Matlab is validated for constrained optimization, and that the

“unconstrained” (in reality constrained with non-binding constraints) bi-level optimizer has been validated previously.

5.2.3 Characterization

With the algorithm verified and guaranteed to work according to the requirements of section 2.3.2, it is possible to characterize its behavior with respect to its different parameters. In particular, it is interesting to see how the performance of the bi-level optimizer is affected when varying the elitism properties and the type of heterogeneity. While many other settings for the genetic algorithms can be varied, these parameters represent the main options introduced by the proposed algorithm. The following characterization is performed on the unconstrained test function with swarms composed of three architectures at most. The first architecture has two design variables, the second architecture has three, and the third architecture has only one. Given that the test function is used, the optimal swarm composition is known as $X^{out} = [1,2,3]$ and the maximum number of agents can be fixed at six to facilitate the convergence process.

Effect of elitism: specific feature of the bi-level optimizer, the elite retention scheme can be activated to see which improvements are possible with elitism. To characterize this effect, the algorithm is run twice on the canonical mission: once without elite retention, and once with elitism activated for 50 percent of the inner loop populations. The recorded data consists of the generations required to attain convergence in the inner loop chromosomes, for each generation of the outer loop. Indeed, the elite retention scheme was designed to accelerate the convergence of the inner loops as the overall algorithm progresses towards convergence. For a given generation of the outer loop, the number of

generations at convergence of each of its inner loop chromosomes is stored in a vector. These results are then averaged, and finally replicated. Indeed, given that the algorithm is stochastic, one thousand replications are used to estimate its variability and obtain robust results mitigating the effects of randomness. Figure 5.44 shows the average of both experiments over the thousand replications and also provides a confidence interval of 95 percent computed from the percentiles of the data. The stopping criteria of the genetic algorithms is fixed at 20 stall generations.

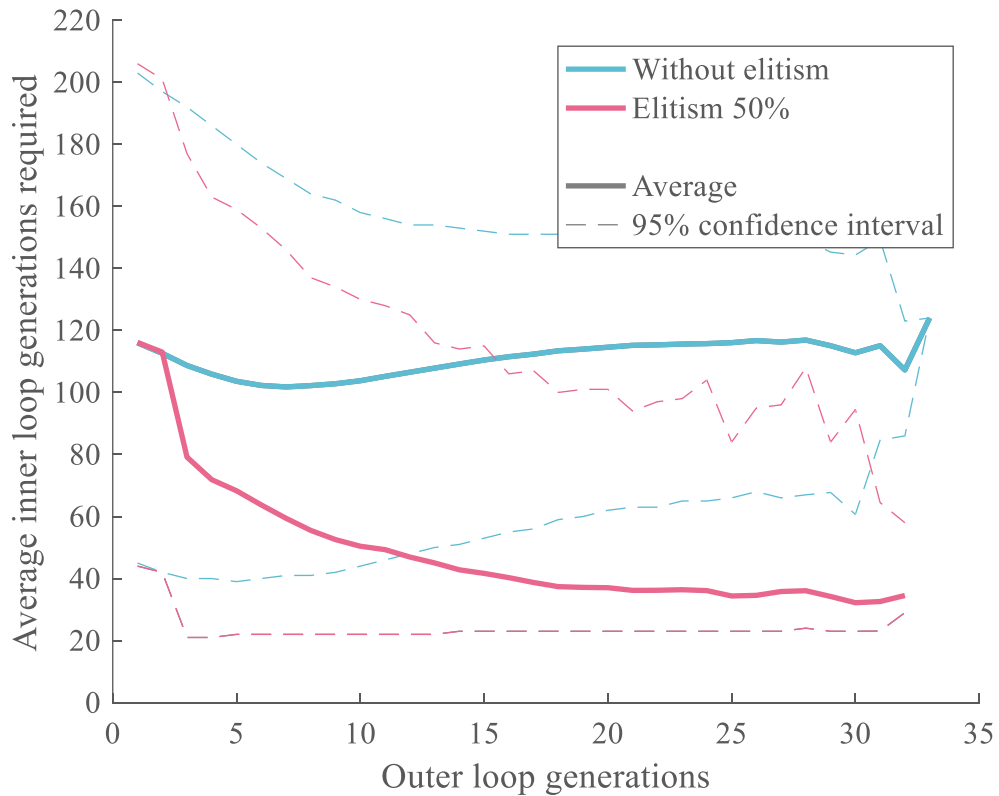


Figure 5.44: Effect of elitism

From Figure 5.44, the number of inner loop generations without elitism remains pretty much constant at around 115 generations on average for the chromosomes of the outer loop. Note that this number seems to slightly decrease between generations one and six, before increasing and settling again at 115. A possible explanation is that in the first generations of the outer loop, the genetic algorithm is still exploring the design space a lot and is trying out swarm configurations that converge a bit more easily. This can also be explained by the fact that variability of the results is more important in the first 10 generations of the outer loop by looking at the confidence interval.

By activating elite retention for half of the population of the inner loops, the number of generations required for their convergence reduces drastically from the first generations of the outer loop. Indeed, the number of required inner generations decreases by 30 percent of its value from the second to the third generation and then goes on to settle at around 35 generations when the outer loop converges. This represents a speedup of around 70 percent for the algorithm. After a couple generations from the outer loop, enough swarm configurations have been considered so that the optimal microscopic configurations of the memory buffer are actually meaningful and untied to specific outer loop configurations. Giving efficient microscopic configurations as starting points for some of the chromosomes of the inner loops accelerates the convergence tremendously for these latter. Then, as the outer loop progresses and starts focusing on efficient swarm compositions, the memory buffer is refined little by little around optimal microscopic configurations which are particularly adapted to the already converging group composition. This helps in continuing to reduce progressively the number of required generations for the inner loops until this number comes very close to the convergence criterion of 20 stall generations.

With the stopping criterion of the algorithms fixed at 20 stall generations, it is expected that the outer loop converges at values a bit above 20 generations. Few instances have the outer loop converge in more than 25 or 30 generations, which explains why the variability of the response increases after 25 generations. Indeed, there are less replications to average the results over. Moreover, it can be seen that with elitism, the lower bound of the confidence interval is at 22 generations for the inner loops (see the lower portion of the 95 percent confidence interval).

Although the average number of inner loop generations are clearly separated by around 80 generations with and without elitism, the confidence intervals exhibit some overlap. In particular, the lower portion of the 95 percent confidence interval without elitism is under the higher portion of the interval with 50 percent of elitism. This indicates that while elitism promises a faster convergence on average, some exceptional cases with elitism might still take equal or superior time to converge than without elitism.

To further quantify the effects of elitism on the behavior of the swarm optimizer, the elite rate is varied in the next experiment.

Effect of elitism fraction: the elitism fraction κ represents the proportion of the initial population of the inner loops which is drawn from the elite memory at each generation of the outer loop. In order to properly capture the effect of a varying elitism rate on the behavior of the algorithm, and especially the outer loop, the convergence criterion is changed from a stalling approach to a precision approach with respect to the expected optimum. If the criterion was left at a given number of stall generations (for instance 20 as for the past experiment), the minimum number of generations required for the outer loop

to converge would be 21 including the initialization. As a consequence, it would not be possible to detect settings incurring a convergence in less than 20 generations. Instead, the genetic algorithms are set to stop once the best fitness of their population is within 1 percent of the theoretical optimum. For this purpose, the verification function described in section 5.2.2.1 is used. Again, the experiment is replicated a thousand times for each elitism rate in order to obtain robust results, shown on Figure 5.45.

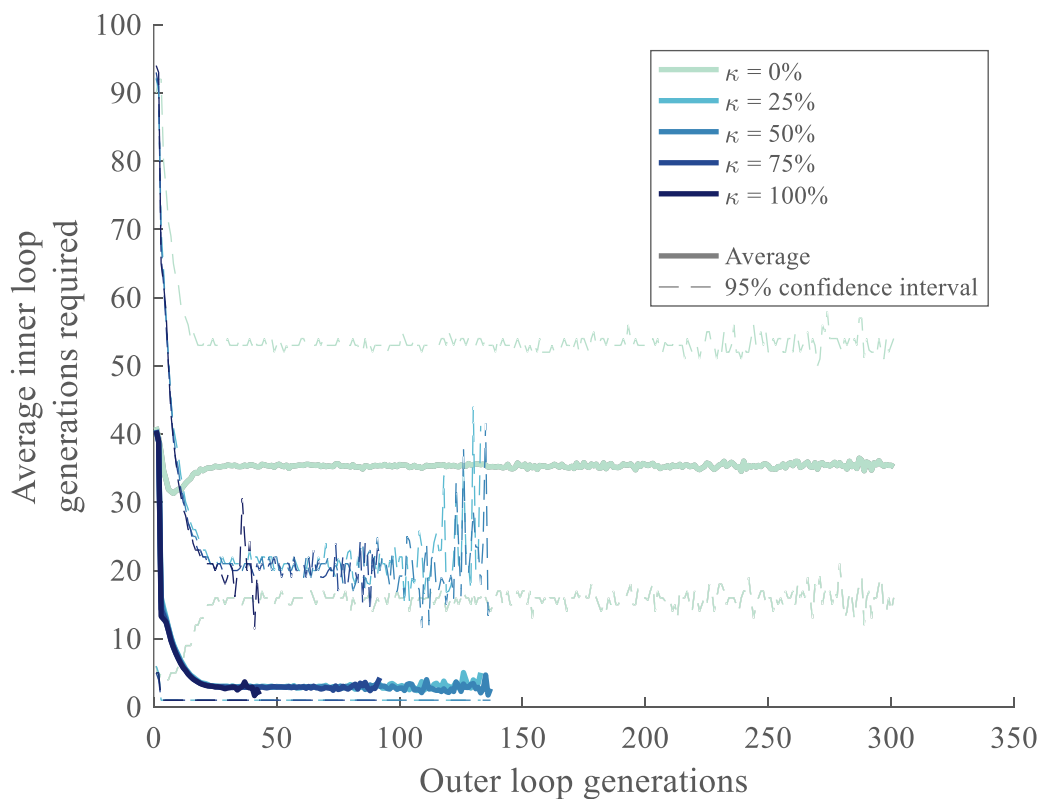


Figure 5.45: Effect of elitism rate

A first observation from Figure 5.45 is that as expected, as long as some elitism rate is applied to the algorithm, this latter requires a lot less iterations for its inner loops to

converge. Indeed, using the new criteria, the number of generations required in the outer loop chromosomes is around 35 without elitism, and 3 with elitism. This represents approximately a 90 percent decrease in the required number of iterations, compared to 70 percent with the other convergence criteria. This is due to the fact that the inner loops do not have to wait for at least 20 stall generations to converge, hence enabling a much lower number of generations for convergence. When there is no elitism, as for the previous experiment, the number of required inner loop generations first decreases during the exploration phase before increasing again and settling around a final value. Moreover, the maximum number of generations of the outer loop is 303 for the precision-based convergence criteria while it was 33 for the stall-based one. Indeed, the one percent precision required to converge is much more constraining than a number of stalling generations.

Regarding the number of generations required for the convergence of the outer loop, the higher the elitism rate, the lesser the number of required generations (see Table 5.14 and Figure 5.46). This corresponds to the requirements of the elite retention scheme: accelerating the convergence of the overall algorithm. Even with low elitism rates, a speedup of at least 50 percent is observed for the convergence of the outer loop and the speedup even reaches 85 percent with 100 percent of elitism.

Table 5.14: Improved convergence with elitism

κ	Maximum number of outer loop generations observed
0	303
0.25	133
0.50	137
0.75	92
1.00	43

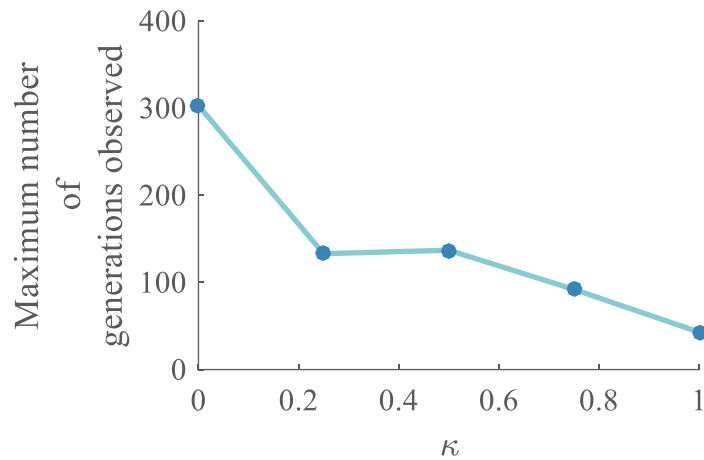


Figure 5.46: Effects of elitism on the outer loop

For each of these measurements, the variability also increases for high numbers of required generations. For instance considering the yellow and red curves, the algorithm converges very often within 75 generations, hence providing a lot of data to average the results from the replications. However, some rare cases converge in more than 75 generations up to around 135 and since these are not averaged with other measurements, their variability is higher. This explains why the end of every colored curve experiences more noise, especially the confidence intervals. Observing the confidence intervals, one sees that the lower portion of the curve without elitism does overlap with the one for the

slowest cases with elitism: on average elitism guarantees a lower number of generations but some rare cases can be slower than without elitism. In addition, one can notice that the minimum number of generations required with elitism is only one: the algorithm converges immediately and is able to give an optimum within one percent of the true optimum in only one generation. Finally, by looking at Figure 5.45, it appears that the elitism rate does not have an effect on the number of generations required for the convergence of the inner loops. However by looking into more detail, Figure 5.47 is obtained.

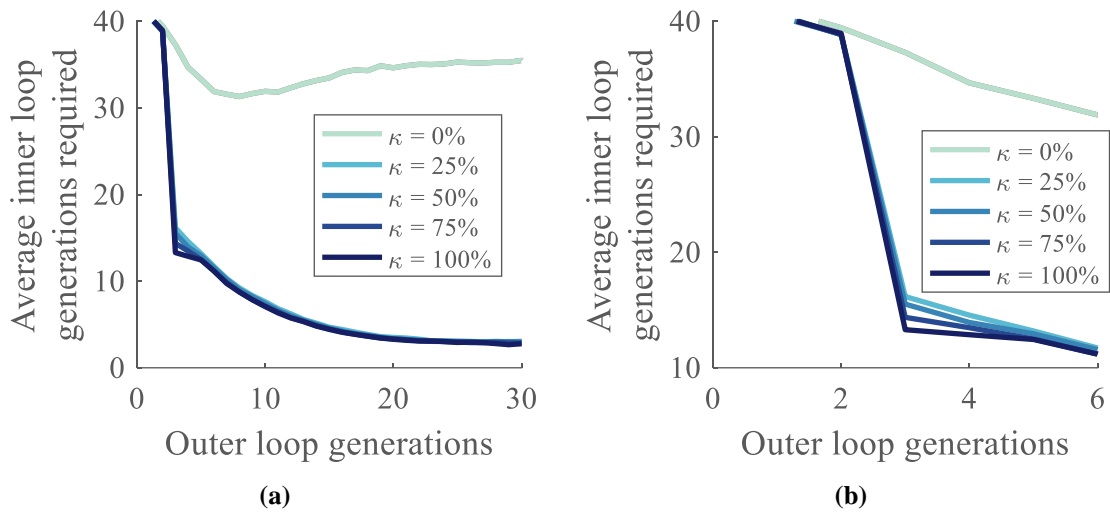


Figure 5.47: Detail on effect of elitism

It shows in particular that the main difference in the response between the elitism rates occurs at the third generation, when there is a sudden drop in the number of generations required for the inner loops. The different values are reported in Table 5.15 and showed on Figure 5.48.

Table 5.15: Main effect of elitism rate

κ	Number of inner generations at 3 rd outer generation
0	37
0.25	16
0.50	15
0.75	14
1.00	13

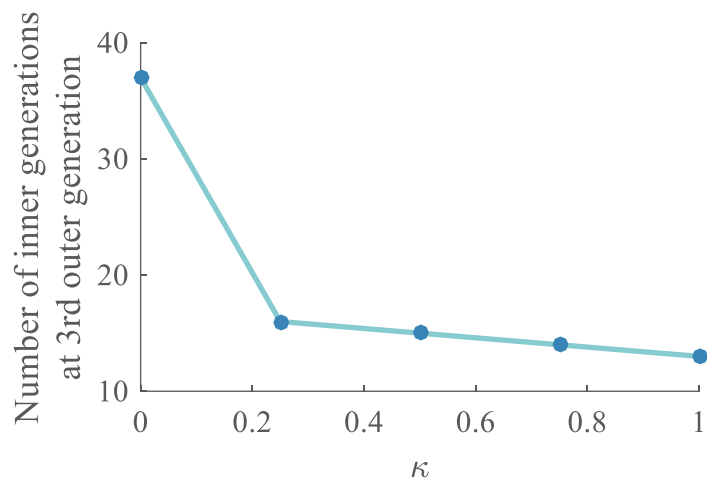


Figure 5.48: Initial effects of elitism

After the drop at the third generation, all curves remain pretty close to each other and do not differ by more than one generation. This confirms the previous observations: the number of iterations required for the convergence of the inner loops converges to a unique value, no matter what the elitism rate is. The order of the curves remains consistent with what is observed on Figure 5.47 (b).

As a conclusion, three main effects of the elitism rate have been established in this paragraph:

- The higher the rate, the fewer generations are required for the convergence of the outer loop (at least 50 percent speedup and up to 85 percent).
- The higher the rate, the fewer generations are required for the convergence of the inner loop (at least 70 percent speedup and up to 90 percent).
- The higher the rate, the bigger the initial drop in required generations for the inner loops (from 57 to 65 percent drop).
- At convergence of the outer loop, the elitism rate does not influence the value of the number of generations required for the convergence of the inner loops (90 percent speedup).

Full vs. partial heterogeneity: final main setting of the bi-level genetic optimizer, the type of heterogeneity is now varied to study its influence on the behavior of the algorithm. The two previous experiments are repeated to capture different effects for each type of convergence: one based on a number of stalled generations, and the other based on the precision of the convergence.

By first setting the convergence condition to 20 stalled generations for both inner and outer loops, a figure comparable to Figure 5.44 is obtained (see Figure 5.49).

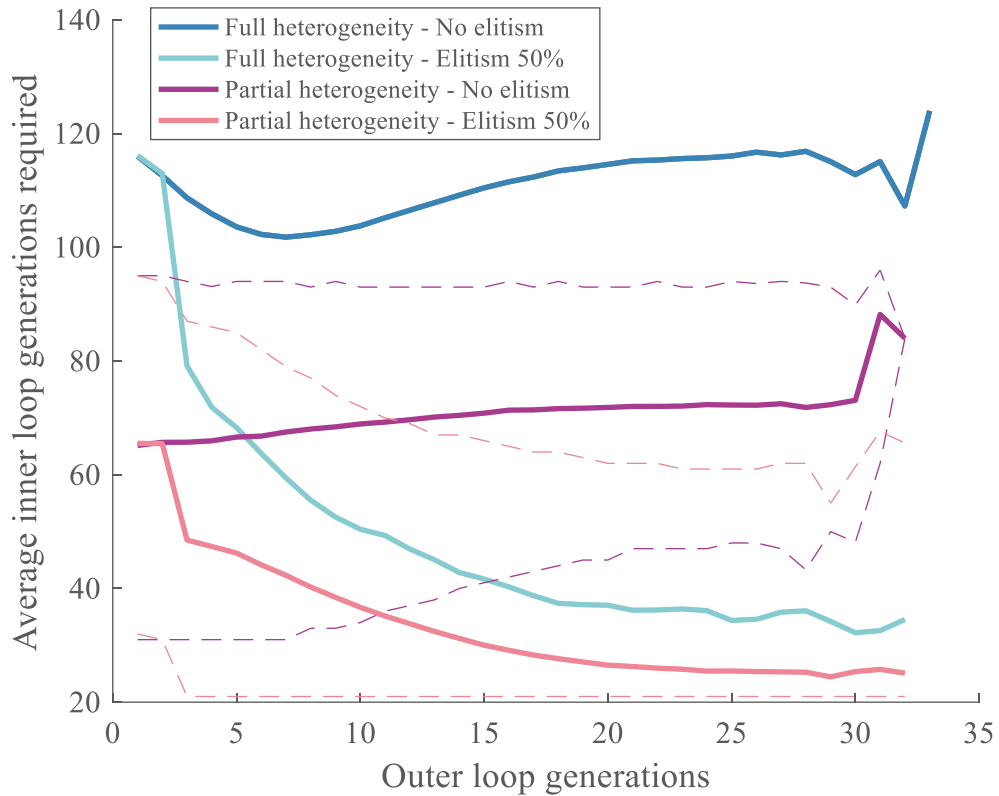


Figure 5.49: Effect of partial heterogeneity and elitism

The main observation is that, as expected and explained earlier on, the convergence for the inner loops is faster with partial heterogeneity when compared to full heterogeneity. This remark is valid with and without elitism. Without elitism, the average number of generations required for the convergence of the inner loops is 111 on average. With partial heterogeneity, this number drops to 71 generations, a reduction of 36 percent. With elitism, the difference is slightly less: 48 against 33 generations, a reduction of 31 percent. This confirms that in both cases (with and without elitism) the convergence of the inner loops is more rapid when considering partial heterogeneity. With full heterogeneity, the difference between with and without elitism is 111 minus 48 generations: a difference of 56 percent. With partial heterogeneity only, this same difference is 71 minus 33: that is 53

percent. Hence partial heterogeneity does not seem to have an impact on the improvements achieved by elitism.

As for the convergence of the outer loop, the number of required generations is fixed at slightly more than 20 generations since this experiment is stall-based. This result is summarized in Figure 5.50 and Table 5.16. Note that the lowest number of generations is 21 which corresponds to the stopping criteria of 20 stalled generations plus the initial generation.

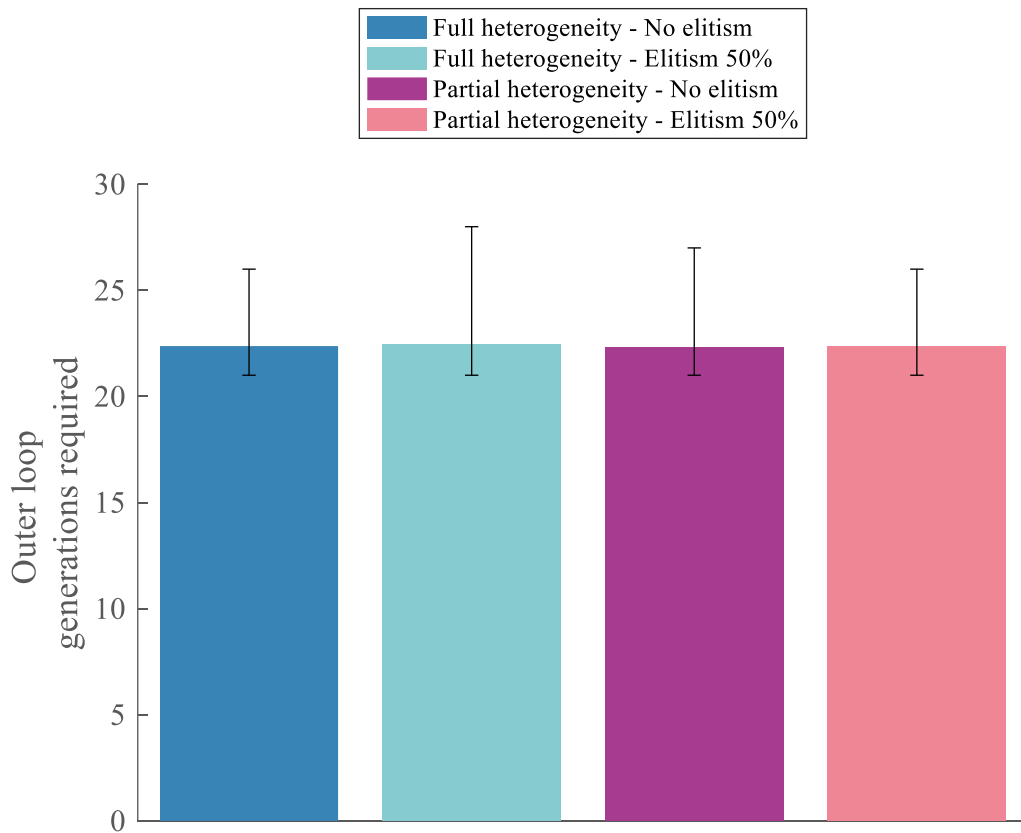


Figure 5.50: Effect of heterogeneity type on stall-based outer loop convergence

Likewise, the convergence criteria can be set to a swarm fitness within one percent of the expected validation value. Figure 5.51 is then obtained in the same fashion as Figure 5.45.

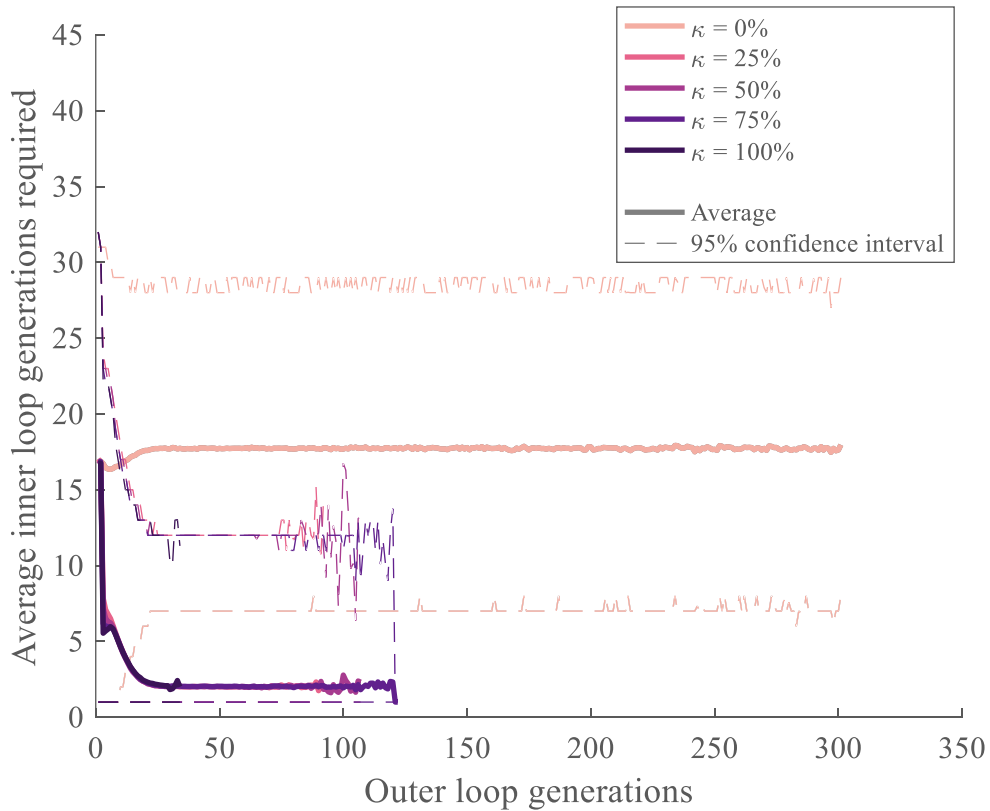


Figure 5.51: Effect of heterogeneity type on stall-based inner loop convergence

This time with partial heterogeneity, activating elitism decreases by 83 percent the number of required generations for the convergence of the inner loops. This number was around 89 percent with full heterogeneity. This is due to the fact that the experiment is reaching the limits of the optimization algorithm. Indeed, the optimizer already performs really well with elitism and full heterogeneity with 4 generations required for convergence.

This number drops to 3 with partial heterogeneity but it would be very difficult to have the algorithm converge in one or two generations. Hence the relative improvement is less with partial heterogeneity since the algorithm is performing better, hence getting closer to convergence limitations. Without elitism, choosing partial heterogeneity incurs a drop of 48 percent in the number of generations, from 35 to 18. With elitism, the drop is around 30 percent on average.

One may notice from Table 5.17 that the average number of inner loop generations required for convergence increases slightly between the 25, 50 percent rates, and the 75, 100 percent elitism rates. Indeed, since the maximum number of outer loop generations decreases with the elitism rate, the average is taken over less many values and tends to privilege the left hand part of the graph when the number of generations for the inner loops is high and still decreasing.

As for the convergence of the outer loop, the results are summarized in Figure 5.52.

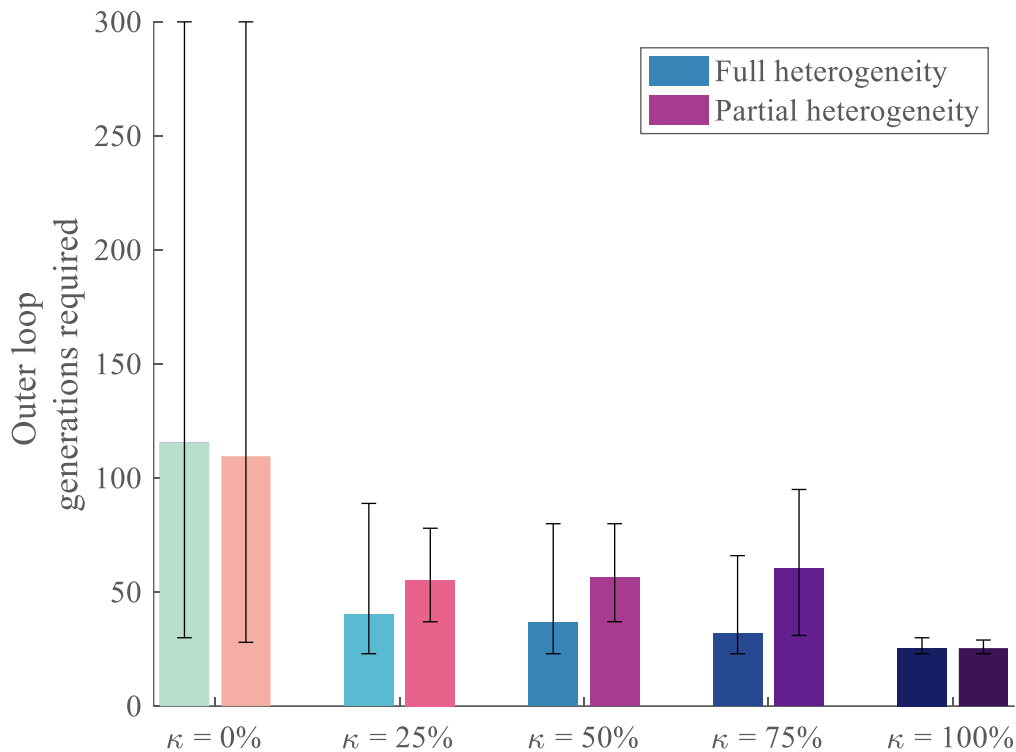


Figure 5.52: Effect of heterogeneity type on precision-based outer loop convergence

These results seem counterintuitive at first since the outer loop seems to take longer to converge with partial heterogeneity than with full heterogeneity which has many more variables. However, after further analysis, one must not forget that the convergence criterion is set at one percent of the actual validation value. With partial heterogeneity, each variable has a much more important weight on the group fitness. The fitness function is then more sensitive to the value of the design variables, making it slightly more unstable than the case with full heterogeneity. In this latter case, more design variables translate into more inertia around a particular design point and it would take a huge change in a design variable to offset the fitness function by a significant value. This is why once the algorithm has started settling around a possible global optimum, it is able to stay around it with more

precision. On the other hand, partial heterogeneity causes more instability around the optimum and the outer loop hence takes more iterations to converge (Figure 5.53).

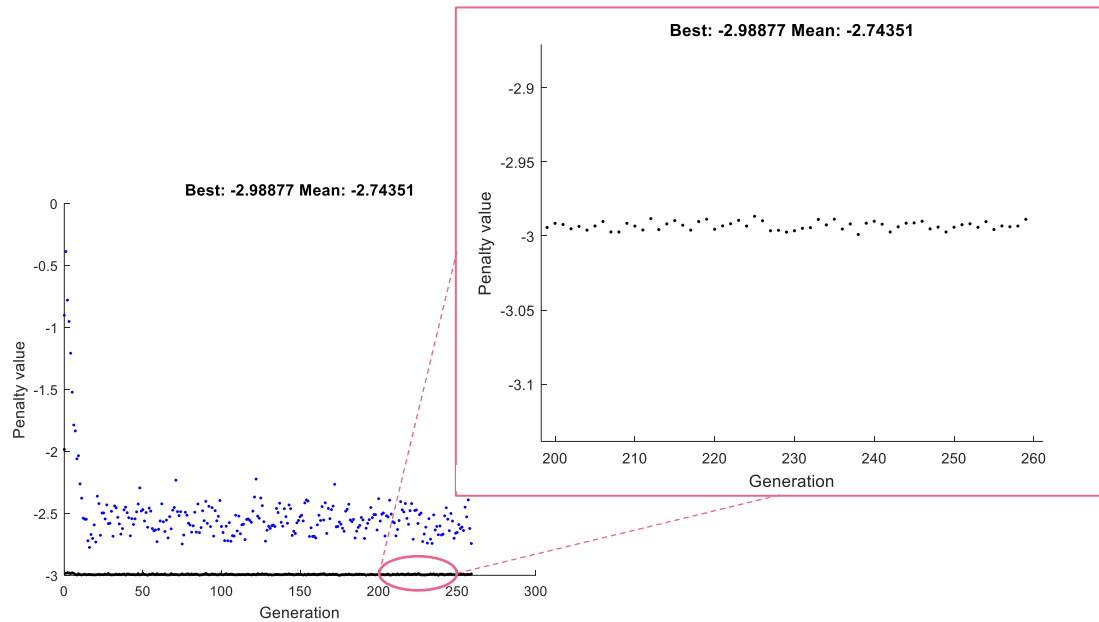


Figure 5.53: Increased convergence instability with partial heterogeneity

It can be seen that the fitness function oscillates a lot above the true optimal value. This does not happen in the stall based approach because it rather considers the average relative change in the best fitness function value instead of the absolute value of the fitness function. It is important to note that in reality, the designer does not know what the true optimum is and will not choose an absolute precision-based convergence criterion.

To conclude, the main influence of the type of heterogeneity on the optimization algorithm has been captured by this experiment:

- With and without elitism, the convergence of the inner loops is more rapid by at least 30 percent when considering partial heterogeneity.
- Partial heterogeneity does not seem to have an impact on the improvements achieved by elitism. Average improvement in number of generations of the inner loop remains around 50 percent.
- When using precision-based convergence, partial heterogeneity tends to make the convergence of the outer loop longer due to a low number of variables and an increased sensitivity/instability. This effect can be removed by relaxing the precision constraint.

In this section, the effect of the principal settings of the optimization algorithm has been studied: elitism, elitism rate, and type of heterogeneity. These effects are summarized in Table 5.16 and Table 5.17. Percentage changes are used in green to show the difference between full and partial heterogeneity. The bold values are the reference used for the percentage changes.

Table 5.16: Optimizer characterization for stall-based convergence

Elitism	Full heterogeneity		Partial heterogeneity	
	Inner loop	Outer loop	Inner loop	Outer loop
No	111	22	71 (-36%)	22
Yes (50%)	48 (-56%)	22	33 (-70%)	22

Table 5.17: Optimizer characterization for precision-based convergence

Elitism rate κ	Full heterogeneity		Partial heterogeneity	
	Inner loop	Outer loop	Inner loop	Outer loop
0	35	115	18 (-48%)	109 (-5%)
0.25	4 (-89%)	40 (-65%)	3 (-91%)	55 (-52%)
0.50	4 (-89%)	36 (-69%)	3 (-91%)	56 (-51%)
0.75	5 (-86%)	32 (-72%)	3 (-91%)	60 (-47%)
1.00	6 (-83%)	25 (-78%)	4 (-89%)	25 (-78%)

CHAPTER 6

CONCLUSION

This final chapter provides a conclusion to this work by presenting a summary of the research methodology, a review of key breakthroughs and contributions, as well as possible perspectives for future research.

6.1 Research summary

The significant progression of the market of drones over the past few years has clearly contributed to an abundant variety of architectures with a wide spectrum of possible applications, mostly in the civil sector. Chapter 1 established a motivation for the present research by pinpointing that although unlocking new capabilities is possible by grouping these architectures into swarms, the design of group robotics systems remains an intricate work in progress. Indeed, multi-robotics present a cost-effective manner to surmount the shortcomings of individual platforms: mostly lack of endurance and limited computational power. Nonetheless, the design of such systems of systems is highly complex: exploiting emergent behaviors, multi-robot systems exhibit stochastic and non-linear relationships which make it extremely difficult to predict the behavior of the overall system from the design variables. Furthermore, a missing link between microscopic level (individual agents) and macroscopic level (group interactions) design spaces was identified in the literature and further complicates the design process. The first chapter hence motivated the need for a holistic approach facilitating the optimal design of multi-robot systems, with a focus on unmanned aerial vehicles.

Chapter 2 ensued with a detailed literature review of existing techniques to identify the missing blocks in three crucial design areas: exploration of the design space (section 5.1), modeling (chapter 4), and optimization (section 5.2). These gaps originated research questions which, after a second literature review leveraging diverse existing methods, enabled the formation of research hypotheses (see Table 6.1).

Table 6.1: Summary of research questions and hypotheses

Area	Gap	Associated research question/hypothesis	Proposed solution for multi-robot systems
Modeling	Missing micro-macro link	RQ 1 (page 92) RQ 2 (page 101)	Global optimization & Mesoscopic modeling
Design space exploration	Extremely large design space	RQ 3.1 (page 132)	Morphological tree with morphological reduction
Optimization	Sequential optimization	RQ 3.2 (page 147)	Bi-level optimization algorithm

The third chapter “Proposed Approach” detailed the implementation choices made to later develop the experiments that would provide response elements with respect to the research questions. In addition, key metrics and main assumptions were established in that chapter so as to clearly define the scope of the research. The following chapters 4 and 5, then respectively focused on the two main parts of the research objective: finding an adequate link between microscopic and macroscopic levels, and multi-level multi-architecture design space exploration.

With chapter 4, hypothesis 1 was first validated as it was confirmed that global optimization can yield significant performance improvements in the design of multi-robot systems. To complete the first observations drawn from the canonical example of section 2.1 page 75, the optimization is performed through two types of sequential optimizers: micro-macro and macro-micro. Both sequential optimizers obtained degraded results when compared to a simultaneous or global optimizer.

In the second half of chapter 4, different types of modeling levels of detail were studied thoroughly and implemented to validate hypothesis 2. The mission used to validate the three models was the rendezvous mission used on the Robotarium system of the Georgia Institute of Technology. A mesoscopic model was constructed step by step as a compromise between a macroscopic model and a microscopic one which were also previously elaborated. The results of a campaign of 40 experiments showed that the mesoscopic model is on average very close to the precision of the microscopic model, and hence representative enough of the performance of the real system. Coupled with the fact that the mesoscopic model is on average five times faster than the microscopic model, this establishes mesoscopic modeling as a good candidate for the design space exploration, and optimization, of multi-level problems.

With an efficient method to link macroscopic and microscopic levels of a robotic swarm established, chapter 5 then focused on the design space exploration portion of the research objective: the generation of alternatives and the optimization of the design of swarms. First, the morphological tree elaborated in chapters 2 and 3 demonstrated promising results in the reduction of design spaces and the organization and representation

of multi-level problems alternatives. In particular, thanks to morphological reduction, the number of alternatives from a real-world problem was reduced by several orders of magnitude, making it more approachable with traditional design space exploration techniques. As an extension to the original work on morphological reduction by [173], its effects were also detailed on multiple-levels design spaces and tradeoff experiments were carried out to establish when the method is beneficial or not.

As a second subsection, the advanced bi-level optimizer designed in chapter 2 and 3 was further detailed, implemented, validated, and characterized. The effects of the elite-retaining scheme were studied for fully heterogeneous and partially heterogeneous swarm configurations. Not only the bi-level optimizer was shown to propose a practical decomposition of the problem without returning to sequential optimization schemes separating the microscopic level and the macroscopic level; but it was also demonstrated that by activating elitism, the inner loop required on average 50% less generations to converge than without elitism.

The overall research process is again summarized on Figure 6.1.

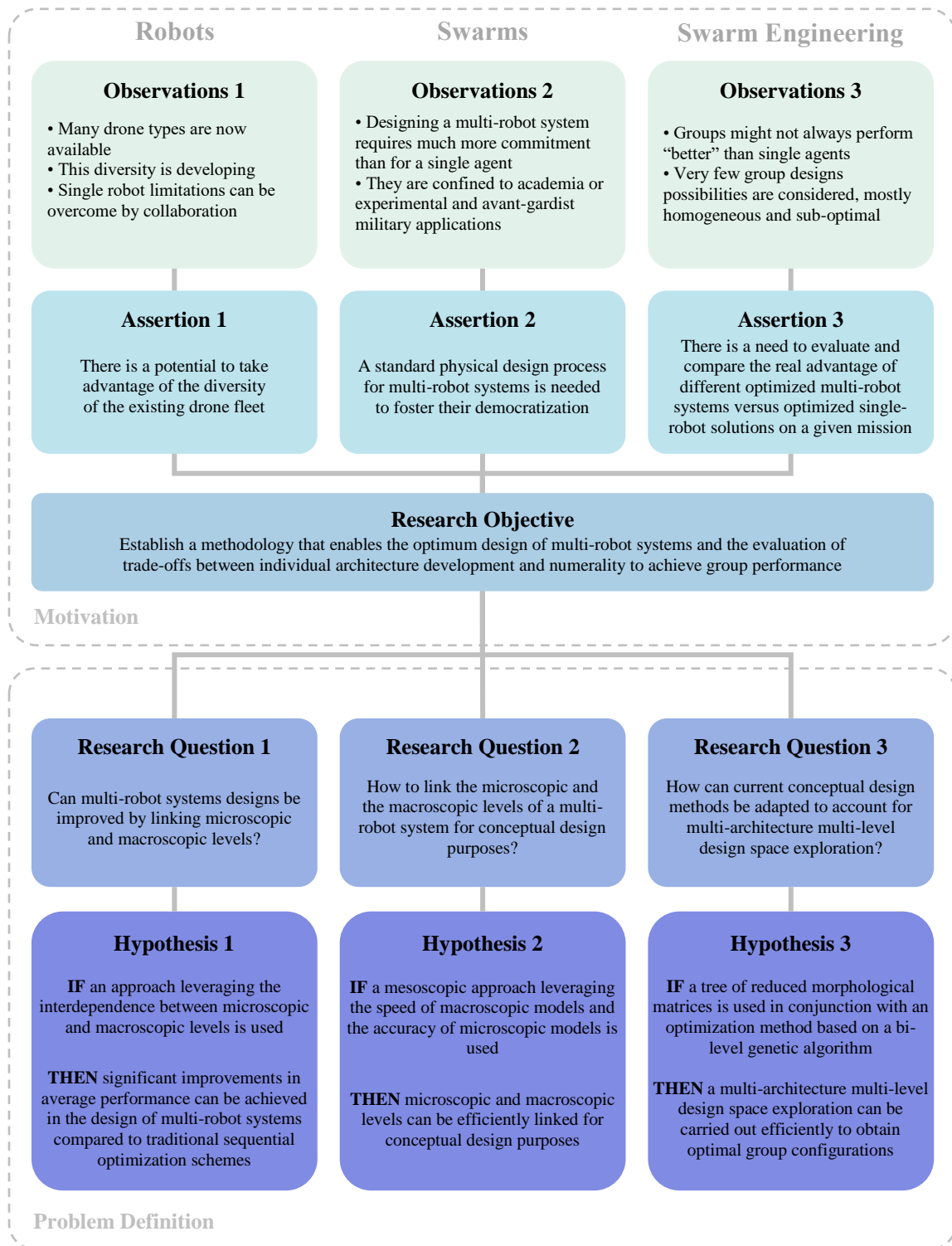


Figure 6.1: Summary of the research process

To conclude with this research summary, the assumptions used throughout the proposed approach have not been found to be inconsistent or incomplete as they were sufficient to validate the different blocks of the methodology. In particular, the reader might notice the correlation between the first and second research questions. Indeed, the first one questions whether a possible micro-macro link could improve the design of swarming systems, while the second one is an attempt to study how this link can be achieved. Hence, an effective link is required to answer the first research question but this link is not accounted for before the second research question. As a result, the second research question was treated first to validate a given modeling methodology which was then used in the first research question to study the benefits of this micro-macro link. However, to ensure a consistent order and articulation of the different parts of the reasoning, the section “Can we improve swarms design?” (RQ1) is presented before the section “How can we improve swarms design?” (RQ2).

6.2 Closing the loop: MASDeM

After answering the research questions and validating each research hypothesis separately with theoretic, conceptual, and benchmarking models, this last chapter demonstrates the use of the whole methodology by putting it all together on an example swarm design exercise. This Multi-Agent Systems Design Methodology is named MASDeM and encompasses all the different steps explained throughout this dissertation and which are summarized here below (see Figure 6.2).

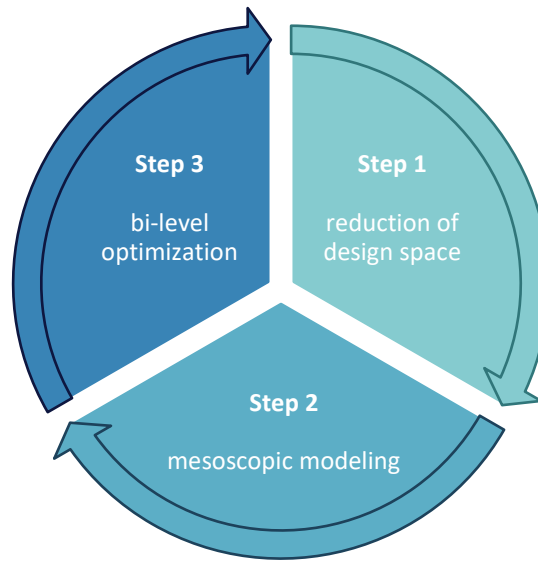


Figure 6.2: Steps of MASDeM

Step 1: reduction of the design space

The designer decides on the first initial set of architectures to be included in the study and documents their morphological matrices as well as the associated technology compatibility matrices. These matrices are then implemented using the object-oriented methodology presented in section 5.1 (see page 292), and assembled into a morphological tree. By using the morphological reduction function of the tree, the designer obtains a reduced number of architectures to be considered while still providing an equivalent coverage of the design space. Note that according to the conclusions of section 5.1, morphological reduction might not prove beneficial for small design spaces with few reducible options. Along with the reduced number of architectures, the design is provided with the list of design variables which need to now be transferred to the modeling part.

Step 2: construction of the mesoscopic model

The second step for the design consists in having a model or simulation of the required mission which can run in a short amount of time while still providing a good level of detail and accuracy so that the design space can be explored efficiently. For this, a mesoscopic model has to be built: a type of model that provides a high level of detail for the constituents of the group but a simplified approach for the group interactions. If pre-existing models are available to the designer, building the mesoscopic model consists mostly in simplifying or relaxing the assumptions of those models and modifying them so that all the variables given by the morphological reduction of step 1 are incorporated. If no pre-existing model of the mission exists, the designer can refer to Figure 4.24 page 274 for examples of levels of details which could constitute a mesoscopic model.

Step 3: optimizing a swarm configuration

Finally, based on the design space definition of step 1, the designer can define the bounds of the variables for all architectures and use the bi-level optimizer to derive an optimal swarm configuration for the required mission. The fitness function used by the optimizer is the mesoscopic model of step 2 which outputs a fitness metric based on the performance metrics of section 3.1 (see page 171). The only task for the designer is to provide the number of variables per architecture, their bounds, and their type. The optimization loops are already implemented in the tool itself.

Step 4 (optional): iteration

Once the optimization algorithm has finished running, or even after a few generations of the outer loop (at least 3 from the observation of section 5.2.3 page 376), the elite buffer contains a representative pool of the most promising architectures for cooperation on the given mission. Hence, it is possible to further reduce the design space by limiting the number of architectures considered by using this elite buffer from step 1 and iterating the process. As a first iteration, the fifty most promising configurations could also be used to decide on better bounds for the design variables and further restrict the size of the design space to be considered. This has the advantage of refining the optimum search over a smaller design space, hence accelerating the overall process. The designer should however be careful not to restrict the design space too much and hence compromise the exploration of different configurations.

In order to close the loop and put all the pieces of the approach back together, a description of the different steps to follow for a real-world problematic was proposed in this subsection. The whole methodology is regrouped under the acronym MASDeM standing for Multi-Agent Systems Design Methodology. The size of the design space is first reduced by using the morphological tree reduction before a mesoscopic model is constructed. Finally, an optimal swarm design can be obtained by applying the bi-level genetic algorithm optimizer on the problem. Without any validation possible with respect to potential similar swarming systems in existence, the performance of the optimal swarm consists in the best known solution so far, and could potentially be shown as competitive when compared to existing single-platform solutions.

6.3 Key contributions

The work carried out for each of the main research axes of the present thesis has led to several crucial observations in the field of multi-robot systems design, some foreseeable and some counterintuitive. These key contributions are recalled, summarized, and listed here below. Additional details can be found when referring to the original corresponding chapters.

Design optimization of multi-robot systems

Sequential vs. simultaneous optimization:

- Currently the research community uses sequential optimization of the different levels, in one order (microscopic then macroscopic is the most common), or the other (macroscopic then microscopic).
- In all the studied cases, simultaneous optimization proved to yield more optimal designs than the ones obtained with sequential optimization.
- Both methods yield similar results only in special cases when interactions (macroscopic level) are neglected between the agents, or when requirements for the different levels are favorable for sequential optimizers. It also depends on the principal axes of the response.
- These improvements range from 1% to 27% on the cost function and average at 16%. However, these improvements are possibly lower bounds with respect to what is achievable on real-world systems.
- Simultaneous optimization is a key enabler for full heterogeneity, a new capability for the design of multi-robot systems.

- The benefits of heterogeneity are mitigated when technology is cheap to acquire with respect to the cost of individual agents.

Efficient optimization of multi-robot systems

- The proposed bi-level optimizer now enables the simultaneous optimization of the macroscopic level and the microscopic level.
- One advantage is that there is no need to implement a different optimizer for each architecture as it was the case for simultaneous optimization. Only two optimizers are required: an inner loop and an outer loop.
- Having an elite retention scheme for the inner loop clearly improves the performance of the algorithm:
 - The higher the elitism rate, the fewer generations are required for the convergence of the outer loop (at least 50% speedup and up to 85%).
 - The higher the elitism rate, the fewer generations are required for the convergence of the inner loop (at least 70% speedup and up to 90%).
 - The higher the elitism rate, the bigger the initial drop in required generations for the inner loops (from 57% to 65% drop).
 - At convergence of the outer loop, the elitism rate does not influence the value of the number of generations required for the convergence of the inner loops (90% speedup).
- The algorithm behaves differently whether heterogeneity is complete (all vehicles of a given architecture are different), or partial (all vehicles of a given architecture are similar):

- With and without elitism, the convergence of the inner loops is more rapid by at least 30 percent when considering partial heterogeneity.
- Partial heterogeneity does not seem to have an impact on the improvements achieved by elitism. The average improvement in number of generations of the inner loop remains around 50 percent.
- When using precision-based convergence, partial heterogeneity tends to make the convergence of the outer loop longer due to a low number of variables and an increased sensitivity/instability. This effect can be removed by relaxing the precision constraint.

Modeling

- Mesoscopic modeling is adapted for exploration and optimization purposes in multi-architecture multi-level design spaces in the early design phases.
- This has been validated on conceptual design phases of multi-robot systems.
- The mesoscopic model always acts as a surrogate of the microscopic model and can be considered as a microscopic model with simplified assumptions.
- On average mesoscopic modeling is more precise than macroscopic modeling while being faster than microscopic modeling.
- A particular application to a consensus multi-robot mission showed that on average:
 - The mesoscopic model ran 5 times faster than the microscopic one.
 - The error of the mesoscopic model on the main performance metric was within 31% of the performance of the real system. A number consistent with conceptual design practices.

Design space exploration for multi-level multi-architecture design spaces

Multi-level morphological reduction

- Applying morphological reduction on multi-level design spaces multiplies the improvements demonstrated in its original application [173].
- Although the higher levels in the hierarchy have most influence on the number of alternatives, it is not clear whether their reduction should be favored to the reduction of lower levels. Indeed, real-world applications tend to show that lower levels are much larger in size than upper levels. Hence removing one option at a top level might have the same effect on the total number of alternatives than removing one at a low level.
- The more options are removed during reduction, the more the impact of the k-factor (number of variables per option) is perceived.
- The number of options per row has more influence on the response than the number of rows in the morphological matrix.
- Morphological reduction is not always beneficial.
- Morphological reduction can be detrimental in problems of high complexity (many variables per option) but demonstrates clear advantages for very large problems as more and more options are removed.
- For extremely large problems, as long as enough options are removed, morphological reduction is always beneficial, quasi-independently of the number of variables per option.

Morphological tree representation

- An object-oriented tree paradigm simplifies the analysis, dynamic manipulation, and representation of multi-level design spaces.
- Complex multi-level operations (computing alternatives, ensuring compatibility, reduction) are now reduced to traversing the morphological tree and calling simple functions on each one of the nodes.
- The morphological tree demonstrated several interesting uses:
 - As a fixed simple bookkeeping tool storing the possible design choices in an organized and graphic fashion.
 - As a fully functional and dynamic design space definition tool used to lock design choices and propagate them through the tree. It can be used to incrementally perform morphological reduction, compute the total of remaining alternatives to study, as well as how many design variables were lost in the process.
 - As an assistant to design optimization by dynamically ensuring that the optimizers consider only feasible designs during the design space exploration.

6.4 Perspectives of future research

By relaxing some of the assumptions used in this research, it is possible to widen the field of application of the proposed methodology and foresee possible extensions.

First, given that the methodology was designed to remain as generic and modular as possible, prospective new applications can be imagined. In particular, the methodology could be applied to the design of product families. Indeed, the dynamic tree structure and

bi-level optimizers can be used to track down requirements, design variables, and performance from one product to the others. The dynamic handling of the outer loop would first enable to derive an optimal number of products in the family without having the designer fix a-priori the size of the group of products. Then, requirements of the macroscopic level (i.e. the market coverage and economics of the product family) would have to flow down to a lower level deriving optimal individual requirements for the products of the family. Hence, a main difference would be that the requirements themselves for the lower levels would be design variables.

Design spaces with more than two levels can also be investigated in more detail with a complete real-world application. One possibility is to go down one more level of detail by adding subsystems of the vehicles as a sub-microscopic level.

Finally, under the more technical aspect of implementation, it could be of interest to study the behavior of the bi-level optimizer when used with other types of optimization algorithms. Indeed, the use of a genetic algorithm was justified in the scope of this research and compared with a full factorial approach, but for other types of applications, mixed-integer programming might prove more appropriate for instance [235].

**A METHODOLOGY TO ACHIEVE MICROSCOPIC/MACROSCOPIC CONFIGURATION
TRADEOFFS IN COOPERATIVE MULTI-ROBOT SYSTEMS DESIGN**

Volume II



A Thesis
Presented to
The Academic Faculty

By

Jean-Guillaume Durand

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Aerospace Engineering

Georgia Institute of Technology
May 2017

Copyright © 2017 by Jean-Guillaume Durand

APPENDIX

LIST OF APPENDICES

APPENDIX A Mathematical derivations	416
A.1 Test function optimum	416
A.1.1 Unconstrained	416
A.1.2 Constrained	422
A.2 Optimization test functions	432
A.2.1 Ackley	432
A.2.2 Dixon-Price	433
A.2.3 Griewank	434
A.2.4 Levy	435
A.2.5 Michalewicz	436
A.2.6 Powell	438
A.2.7 Rastrigin	438
A.2.8 Rosenbrock	439
A.2.9 Rotated hyper-ellipsoid	440
A.2.10 Schwefel	441
A.2.11 Sphere	442
A.2.12 Styblinski-Tang	443
A.2.13 Sum squares	444
A.3 Barrier certificates formulation	445
APPENDIX B Matlab code	449
B.1 Modeling	449

B.1.1	Macroscopic	449
B.1.2	Mesoscopic	450
B.1.3	Microscopic.....	464
B.1.4	Real system	481
B.1.5	Scripts	489
B.2	Optimization	493
B.2.1	Sequential and simultaneous optimization.....	493
B.2.2	Bi-level optimizer	504
B.2.3	Unit tests	512
B.2.4	Plots for the test function	522
B.2.5	Plots optimizer	539
B.3	Design space exploration	555
B.3.1	Classes.....	555
B.3.2	Unit tests	565
B.3.3	Example scripts.....	572
B.3.4	Interfaces.....	576
B.3.5	PACE cluster scripts	595
B.3.6	Plots.....	604
APPENDIX C ROS/GAZEBO files.....		622
C.1	Gritbot URDF description file	623
C.2	Robotarium world file.....	624
C.3	Consensus mission launch file	626
C.4	Navigation package.....	629

C.4.1	Consensus	629
C.4.2	Tracker	637
C.4.3	Logger	639

LIST OF APPENDIX TABLES

Table A-1: KKT analysis for inner loop constraints.....	426
Table A-2: KKT analysis for outer loop constraints.....	430
Table A-3: Optimum of the Michalewicz test function	436

LIST OF APPENDIX FIGURES

Figure A-1: Constraints on the first design variable.....	427
Figure A-2: Constraints on the second design variable	428
Figure A-3: Constraints on two design variables.....	428
Figure A-4: Ackley function representation	432
Figure A-5: Dixon-Price function representation	433
Figure A-6: Griewank function representation over a varied range	434
Figure A-7: Levy function representation.....	435
Figure A-8: Michalewicz function representation	437
Figure A-9: Local representation of the Michalewicz function.....	437
Figure A-10: Rastrigin function representation	438
Figure A-11: Rosenbrock function representation.....	439
Figure A-12: Rotated Hyper-Ellipsoid function representation.....	440
Figure A-13: Schwefel function representation	441
Figure A-14: Sphere function representation.....	442
Figure A-15: Styblinski-Tang function representation	443
Figure A-16: Sum of squares function representation	444

LIST OF APPENDIX EQUATIONS

Equation A-1: Unconstrained verification function for the optimizer	416
Equation A-2: Gradient of the verification function	418
Equation A-3: Hessian definition.....	419
Equation A-4: Hessian matrix for the verification function	420
Equation A-5: Hessian matrix evaluated at the stationary point.....	420
Equation A-6: Unconstrained global optimum	421
Equation A-7: Minimum value of the test function at the unconstrained optimum	421
Equation A-8: Constrained outer loop optimization problem.....	422
Equation A-9: Constrained inner loop optimization problem.....	422
Equation A-10: Constrained optimization problem	423
Equation A-11: Standard form for the constrained optimization problem	423
Equation A-12: Lagrangian of the verification optimization problem	424
Equation A-13: Lagrangian derivative with respect to the inner loop design vector	424
Equation A-14: Lagrangian derivative with respect to the inner loop design vector	425
Equation A-15: KKT conditions for inner loop constraints.....	425
Equation A-16: Derivation of the inner loop constrained optimum	427
Equation A-17: KKT conditions for the outer loop	429
Equation A-18: Matricial form of inter-robot collision constraints.....	446
Equation A-19: World boundaries constraints.....	447
Equation A-20: Matricial form of world boundaries constraints.....	447
Equation A-21: Quadratic program based controller	448

APPENDIX A

MATHEMATICAL DERIVATIONS

A.1 *Test function optimum*

It was stated in section 5.2.2.1 (see page 364) that the verification function f is designed to have known specific optimum value and locations. However, this was not proven. This appendix hence provides the analytical proof for the minimal value of the fitness function used for the verification of the bi-level optimization algorithm.

A.1.1 Unconstrained

The unconstrained verification function is defined as follow:

Equation A-1: Unconstrained verification function for the optimizer

$$f(X = [X^{in}, X^{out}]) = \underbrace{\left(\sum_{i=1}^{N_{vars}(X^{in})} [X_i^{in} - x_i^*]^2 \right)}_{f_{in}(X^{in})} - \underbrace{\left(\sum_{i=1}^{N_{arch}} \frac{1}{1 + [X_i^{out} - i]^2} \right)}_{f_{out}(X^{out})}$$

With $N_{vars}(X^{in})$ the number of variables in the inner loop design vector, N_{archs} the number of variables in the outer loop design vector (or equivalently the number of architectures considered). The value of x_i^* was given in Equation 5.12 page 368. The design vector X is partitioned between inner loop and outer loop design variables as $X = [X^{in}, X^{out}]$.

Finding the minimum of this fitness function consists in computing the stationary points thanks to the gradient, and making sure that the Hessian matrix is positive semi-definite around these points before comparing them to find the global optimum. The first step is computing the gradient $\nabla f = \frac{df}{dX} = \left[\frac{\partial f}{\partial X^{in}}, \frac{\partial f}{\partial X^{out}} \right]$. One can first note that $\frac{\partial f}{\partial X^{in}} = \frac{\partial(f_{in}+f_{out})}{\partial X^{in}} = \frac{df_{in}}{dX^{in}} + \underbrace{\frac{\partial f_{out}}{\partial X^{in}}}_0 = \frac{df_{in}}{dX^{in}}$ since f_{out} does not depend on X^{in} . Similarly, $\frac{\partial f}{\partial X^{out}} =$

$\frac{df_{out}}{dX^{out}}$. Each of these terms is then considered separately here after.

The first term is $\frac{df_{in}}{dX^{in}} = \left[\frac{\partial f_{in}}{\partial X_1^{in}}, \frac{\partial f_{in}}{\partial X_2^{in}}, \dots, \frac{\partial f_{in}}{\partial X_{Nvars(X^{in})}^{in}} \right] = \left(\frac{\partial f_{in}}{\partial X_i^{in}} \right)_{i \in \llbracket 1, Nvars(X^{in}) \rrbracket}$ and

each $\frac{\partial f_{in}}{\partial X_i^{in}}$ can be computed as $\frac{\partial f_{in}}{\partial X_i^{in}} = \frac{\partial}{\partial X_i^{in}} \left(\sum_{k=1}^{Nvars(X^{in})} [X_k^{in} - x_k^*]^2 \right) = 2(X_i^{in} - x_i^*)$.

The second term is $\frac{df_{out}}{dX^{out}} = \left(\frac{\partial f_{out}}{\partial X_i^{out}} \right)_{i \in \llbracket 1, N_{arch} \rrbracket}$ and each of the $\frac{\partial f_{out}}{\partial X_i^{out}}$'s can be written

as $\frac{\partial f_{out}}{\partial X_i^{out}} = \frac{\partial}{\partial X_i^{out}} \left(\sum_{k=1}^{N_{archs}} \frac{1}{1+[X_k^{out}-k]^2} \right) = \frac{2(X_i^{out}-i)}{[1+(X_i^{out}-i)^2]^2}$.

Putting it all back together, the gradient of the fitness function f is expressed as:

Equation A-2: Gradient of the verification function

$$\nabla f(X) = \left\{ \begin{array}{l} 2(X_1^{in} - x_1^*) \\ 2(X_2^{in} - x_2^*) \\ \vdots \\ 2(X_{N_{vars}(X^{in})}^{in} - x_{N_{vars}(X^{in})}^*) \\ \hline \frac{2(X_1^{out} - 1)}{[1 + (X_1^{out} - 1)^2]^2} \\ \frac{2(X_2^{out} - 2)}{[1 + (X_2^{out} - 2)^2]^2} \\ \vdots \\ \frac{2(X_{N_{arch}}^{out} - N_{arch})}{[1 + (X_{N_{arch}}^{out} - N_{arch})^2]^2} \end{array} \right\}$$

Setting $\nabla f = \vec{0}$ to find the stationary points, one finds that the inner loop elements are set to zero if and only if $X_i^{in} = x_i^*, \forall i \in \llbracket 1, N_{vars}(X^{in}) \rrbracket$ and that the outer loop elements are null if and only if $X_i^{out} = i, \forall i \in \llbracket 1, N_{arch} \rrbracket$. Hence, there is only one stationary point. Finally, one must verify that this point actually corresponds to a minimum. One possible way to do that is to ensure that the verification function f is locally convex around this stationary point, which is equivalent to having a positive semi-definite Hessian matrix. The Hessian of f is defined as:

Equation A-3: Hessian definition

$$H_f = \left(\frac{\partial^2 f}{\partial X_i \partial X_j} \right)_{i,j} = \left(\frac{\partial \nabla f_j}{\partial X_i} \right)_{i,j} = \left(\frac{\partial \nabla f}{\partial X_i} \right)_{i,j} = \begin{bmatrix} \frac{\partial \nabla f}{\partial X_1^{in}} \\ \frac{\partial \nabla f}{\partial X_2^{in}} \\ \vdots \\ \frac{\partial \nabla f}{\partial X_{N_{vars}(X^{in})}^{in}} \\ \frac{\partial \nabla f}{\partial X_1^{out}} \\ \frac{\partial \nabla f}{\partial X_2^{out}} \\ \vdots \\ \frac{\partial \nabla f}{\partial X_{N_{arch}}^{out}} \end{bmatrix}$$

By looking at the shape of the gradient given in Equation A-2, there is only one component of this gradient that depends on a given variable X_i of the design vector X (be it an inner loop design variable X_i^{in} or an outer loop variable X_i^{out}). For a given design variable X_i , the only component for which the derivative of the gradient will not be zero is ∇f_i . Since the form of the Hessian is $H_f = \left(\frac{\partial \nabla f_j}{\partial X_i} \right)_{i,j}$, this means that all non-diagonal terms of the Hessian matrix are zero. As for the diagonal terms, two cases have to be considered: the ones depending on inner loop variables, and the one depending on outer loop variables. In the first case, the terms are $\frac{\partial \nabla_i^{in} f}{\partial X_i^{in}} = \frac{\partial}{\partial X_i^{in}} [2(X_i^{in} - x_i^*)] = 2$. For the outer loop terms, $\frac{\partial \nabla_i^{out} f}{\partial X_i^{out}} = \frac{\partial}{\partial X_i^{out}} \left(\frac{2(X_i^{out} - i)}{[1+(X_i^{out} - i)^2]^2} \right) = \frac{2[1-3(X_i^{out} - i)^2]}{[1+(X_i^{out} - i)^2]^3}$. Note that $\nabla_i^{in} f$ and $\nabla_i^{out} f$ refer to terms of the gradient depending on respectively inner variable i with $i \in \llbracket 1, N_{vars}(X^{in}) \rrbracket$, and outer variable i with $i \in \llbracket 1, N_{arch} \rrbracket$. As a consequence, the Hessian has the following form:

Equation A-4: Hessian matrix for the verification function

$$H_f(X) = \begin{bmatrix}
 \begin{matrix} \color{blue}{\overbrace{2 & 0 & \dots & 0}^{N_{vars}(X^{in})}} \\ \color{blue}{\underbrace{0 & 2 & \dots & 0}_{N_{vars}(X^{in})}} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 2 \end{matrix} & 0 & 0 & \dots & 0 \\
 0 & \frac{2[1-3(X_1^{out}-1)^2]}{[1+(X_1^{out}-1)^2]^3} & 0 & \dots & 0 \\
 0 & 0 & \frac{2[1-3(X_2^{out}-2)^2]}{[1+(X_2^{out}-2)^2]^3} & \dots & 0 \\
 \vdots & \vdots & \vdots & \ddots & \vdots \\
 0 & 0 & 0 & \dots & \frac{2[1-3(X_{N_{archs}}^{out}-N_{archs})^2]}{[1+(X_{N_{archs}}^{out}-N_{archs})^2]^3}
 \end{bmatrix}$$

N_{arch}

The diagonal terms linked with the outer loop variables have the value 2 when

evaluated at the stationary point, indeed $\frac{2[1-3(X_i^{out}-i)^2]}{[1+(X_i^{out}-i)^2]^3} = 2 \frac{[1-3(i-i)^2]}{[1+(i-i)^2]^3} = 2 \times \frac{1}{1} = 2$. Hence

the Hessian at the stationary point corresponds to:

Equation A-5: Hessian matrix evaluated at the stationary point

$$H_f(X^*) = \begin{bmatrix}
 \begin{matrix} \color{blue}{\overbrace{2 & 0 & \dots & 0}^{N_{vars}(X^{in})}} \\ \color{blue}{\underbrace{0 & 2 & \dots & 0}_{N_{vars}(X^{in})}} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 2 \end{matrix} & 0 & 0 & \dots & 0 \\
 0 & 2 & 0 & \dots & 0 \\
 0 & 0 & 2 & \dots & 0 \\
 \vdots & \vdots & \vdots & \ddots & \vdots \\
 0 & 0 & 0 & \dots & 2
 \end{bmatrix} = 2 \times I$$

With I the identity matrix having the size corresponding to the number of elements in the complete design vector X . One can then conclude that the Hessian at the stationary point is positive semi-definite. This proves that the verification function is locally convex

around the stationary point and hence that the stationary point derived earlier on is indeed a local minimum of the verification fitness function. Given that it is the only stationary point, this also sufficiently proves that it is the global minimum:

Equation A-6: Unconstrained global optimum

$$X_i^{*in} = x_i^* \forall i \in \llbracket 1, N_{vars}(X^{in}) \rrbracket$$

$$X_i^{*out} = i \forall i \in \llbracket 1, N_{arch} \rrbracket$$

The value of the fitness function at the minimum is hence:

Equation A-7: Minimum value of the test function at the unconstrained optimum

$$\begin{aligned} f(X^*) &= \left(\sum_{i=1}^{N_{vars}(X^{in})} [X_i^{*in} - x_i^*]^2 \right) - \left(\sum_{i=1}^{N_{arch}} \frac{1}{1 + [X_i^{*out} - i]^2} \right) \\ &= \underbrace{\left(\sum_{i=1}^{N_{vars}(X^{in})} [x_i^* - x_i^*]^2 \right)}_0 - \left(\sum_{i=1}^{N_{arch}} \frac{1}{1 + [i - i]^2} \right) \\ &= - \sum_{i=1}^{N_{arch}} 1 \\ &= -N_{arch} \end{aligned}$$

To conclude, the minimum of the verification fitness function was analytically derived and corresponds to what was announced in section 5.2.2.1.

A.1.2 Constrained

The constrained optimization problems for the inner and outer loops were described in equations Equation 5.15 and Equation 5.16 and are recalled here:

Equation A-8: Constrained outer loop optimization problem

$$\begin{aligned} & \min_{X^{out}} in(X^{out}) \\ & \text{subject to} \\ & \left\{ \begin{array}{l} X_i^{out} \geq X_i^{*out} + 1, \forall i \in \llbracket 1, N_{arch} \rrbracket \\ 1 \leq \sum_{i=1}^{N_{arch}} X_i^{out} \leq N_{max} \\ 0 \leq X_i^{out} \leq N_{max}, \forall i \in \llbracket 1, N_{arch} \rrbracket \\ X_i^{out} \in \mathbb{N}, \forall i \in \llbracket 1, N_{arch} \rrbracket \end{array} \right. \end{aligned}$$

Equation A-9: Constrained inner loop optimization problem

$$\begin{aligned} & \min_{X^{in}} f(X^{in}) \\ & \text{subject to} \\ & \left\{ \begin{array}{l} X_i^{in} \geq X_i^{*in} + 1, \forall i \in \llbracket 1, N_{vars}(X^{in}) \rrbracket \\ l_b \leq X^{in} \leq u_b \end{array} \right. \end{aligned}$$

The mathematical formulation is able to consider both inner and outer loops at the same time since there are no dynamic size allocation issues for the design vectors of the inner loop. The optimization problem can hence be written as:

Equation A-10: Constrained optimization problem

$$\begin{aligned}
 & \min_X f(X) \\
 & \text{subject to} \\
 & \left\{ \begin{array}{l}
 X_i^{out} \geq X_i^{*out} + 1, \forall i \in \llbracket 1, N_{arch} \rrbracket \\
 1 \leq \sum_{i=1}^{N_{arch}} X_i^{out} \leq N_{max} \\
 0 \leq X_i^{out} \leq N_{max}, \forall i \in \llbracket 1, N_{arch} \rrbracket \\
 X_i^{in} \geq X_i^{*in} + 1, \forall i \in \llbracket 1, N_{vars}(X^{in}) \rrbracket \\
 l_b \leq X^{in} \leq u_b \\
 X_i^{out} \in \mathbb{N}, \forall i \in \llbracket 1, N_{arch} \rrbracket
 \end{array} \right.
 \end{aligned}$$

Putting it in standard form, Equation A-11 is obtained:

Equation A-11: Standard form for the constrained optimization problem

$$\begin{aligned}
 & \min_X f(X) \\
 & \text{subject to} \\
 & \left\{ \begin{array}{l}
 X_i^{*in} + 1 - X_i^{in} \leq 0, \forall i \in \llbracket 1, N_{vars}(X^{in}) \rrbracket \\
 l_{b_i} - X_i^{in} \leq 0, \forall i \in \llbracket 1, N_{vars}(X^{in}) \rrbracket \\
 X_i^{in} - u_{b_i} \leq 0, \forall i \in \llbracket 1, N_{vars}(X^{in}) \rrbracket \\
 \\
 X_i^{*out} + 1 - X_i^{out} \leq 0, \forall i \in \llbracket 1, N_{arch} \rrbracket \\
 1 - \sum_{i=1}^{N_{arch}} X_i^{out} \leq 0 \\
 \sum_{i=1}^{N_{arch}} X_i^{out} - N_{max} \leq 0 \\
 -X_i^{out} \leq 0, \forall i \in \llbracket 1, N_{arch} \rrbracket \\
 X_i^{out} - N_{max} \leq 0, \forall i \in \llbracket 1, N_{arch} \rrbracket \\
 X_i^{out} \in \mathbb{N}, \forall i \in \llbracket 1, N_{arch} \rrbracket
 \end{array} \right.
 \end{aligned}$$

The first step of this analysis is to compute the Lagrangian of the problem (see Equation A-12).

Equation A-12: Lagrangian of the verification optimization problem

$$\begin{aligned} \mathcal{L}(X) = f(X) &+ \sum_{i=1}^{N_{vars}(X^{in})} \alpha_i (X_i^{*in} + 1 - X_i^{in}) + \sum_{i=1}^{N_{vars}(X^{in})} \beta_i (l_{b_i} - X_i^{in}) + \sum_{i=1}^{N_{vars}(X^{in})} \gamma_i (X_i^{in} - u_{b_i}) \\ &+ \sum_{i=1}^{N_{arch}} \delta_i (X_i^{*out} + 1 - X_i^{out}) + \epsilon \left(1 - \sum_{i=1}^{N_{arch}} X_i^{out} \right) + \zeta \left(\sum_{i=1}^{N_{arch}} X_i^{out} - N_{max} \right) \\ &+ \sum_{i=1}^{N_{arch}} \eta_i (-X_i^{out}) + \sum_{i=1}^{N_{arch}} \theta_i (X_i^{out} - N_{max}) \end{aligned}$$

The coefficients $\alpha, \beta, \gamma, \delta, \epsilon, \zeta, \eta$, and θ are the Lagrange multipliers associated with the different inequality constraints. The next step is now to compute the gradient of the Lagrangian $\nabla \mathcal{L}$ with respect to the design vector X and with respect to the Lagrange multipliers: $\nabla \mathcal{L} = \left[\frac{\partial \mathcal{L}}{\partial X}, \frac{\partial \mathcal{L}}{\partial \alpha}, \frac{\partial \mathcal{L}}{\partial \beta}, \frac{\partial \mathcal{L}}{\partial \gamma}, \frac{\partial \mathcal{L}}{\partial \delta}, \frac{\partial \mathcal{L}}{\partial \epsilon}, \frac{\partial \mathcal{L}}{\partial \zeta}, \frac{\partial \mathcal{L}}{\partial \eta}, \frac{\partial \mathcal{L}}{\partial \theta} \right]$. Similarly to the previous section, the gradient element $\frac{\partial \mathcal{L}}{\partial X}$ can also be decomposed into components linked with inner variables $\frac{\partial \mathcal{L}}{\partial X_i^{in}}$, and components related to outer loop variables $\frac{\partial \mathcal{L}}{\partial X_i^{out}}$. Computing these terms separately, the first one is:

Equation A-13: Lagrangian derivative with respect to the inner loop design vector

$$\frac{\partial \mathcal{L}}{\partial X_i^{in}} = 2(X_i^{in} - x_i^*) - \alpha_i - \beta_i + \gamma_i$$

And the second one is:

Equation A-14: Lagrangian derivative with respect to the inner loop design vector

$$\frac{\partial \mathcal{L}}{\partial X_i^{out}} = \frac{2(X_i^{out} - i)}{[1 + (X_i^{out} - i)^2]^2} - \delta_i - \epsilon + \zeta - \eta_i + \theta_i$$

The following two paragraphs detail the analysis by separating inner loop constraints from outer loop constraints.

Inner loop constraints: the first-order necessary conditions, also known as Karush–Kuhn–Tucker (KKT) conditions to be solved for the inner loop constraints are thus for all i in $\llbracket 1, N_{vars}(X^{in}) \rrbracket$:

Equation A-15: KKT conditions for inner loop constraints

$$\begin{cases} 2(X_i^{in} - x_i^*) - \alpha_i - \beta_i + \gamma_i = 0 \\ \alpha_i(X_i^{*in} + 1 - X_i^{in}) = 0 \\ \beta_i(l_{b_i} - X_i^{in}) = 0 \\ \gamma_i(X_i^{in} - u_{b_i}) = 0 \end{cases}$$

To study whether the constraints are active or not, the cases where the Lagrange multipliers are zero are first studied. The analysis is summarized in Table A-1.

Table A-1: KKT analysis for inner loop constraints

$\beta_i \neq 0$	$\Rightarrow l_{b_i} - X_i^{in} = 0 \Rightarrow X_i^{in} = l_{b_i}$ Impossible as the constraint $X_i^{in} \geq X_i^{*in} + 1 > l_{b_i}$ would be violated.
$\alpha_i \neq 0$	$\Rightarrow \begin{cases} X_i^{in} - u_{b_i} = 0 \\ X_i^{*in} + 1 - X_i^{in} = 0 \end{cases} \Rightarrow \begin{cases} X_i^{in} = u_{b_i} \\ X_i^{in} = X_i^{*in} + 1 \end{cases}$ Impossible since u_{b_i} is set so that $u_{b_i} \neq X_i^{*in} + 1$
$\gamma_i \neq 0$	$\Rightarrow \begin{cases} X_i^{in} - u_{b_i} = 0 \\ 2(X_i^{in} - x_i^*) + \gamma_i = 0 \end{cases}$
$\alpha_i = 0$	$\Rightarrow \begin{cases} X_i^{in} = u_{b_i} \\ \gamma_i = -2(X_i^{in} - x_i^*) \end{cases}$ $\Rightarrow \begin{cases} X_i^{in} = u_{b_i} \\ \gamma_i = -2(u_{b_i} - x_i^*) \end{cases}$
$\beta_i = 0$	
$\alpha_i \neq 0$	$\Rightarrow \begin{cases} X_i^{*in} + 1 - X_i^{in} = 0 \\ 2(X_i^{in} - x_i^*) - \alpha_i = 0 \end{cases}$ $\Rightarrow \begin{cases} X_i^{in} = X_i^{*in} + 1 \\ \alpha_i = 2(X_i^{in} - x_i^*) \end{cases}$
$\gamma_i = 0$	$\Rightarrow \begin{cases} X_i^{in} = X_i^{*in} + 1 \\ \alpha_i = 2 \end{cases}$
$\alpha_i = 0$	$\Rightarrow 2(X_i^{in} - x_i^*) = 0 \Rightarrow X_i^{in} = x_i^*$ Impossible as the constraint $X_i^{in} \geq X_i^{*in} + 1$ would be violated. Note that $X_i^{*in} = x_i^*$.

Looking at Table A-1, there are two solutions: $\{X_i^{in}, \alpha_i, \beta_i, \gamma_i\} = \{u_{b_i}, 0, 0, -2(u_{b_i} - x_i^*)\}$ and $\{X_i^{in}, \alpha_i, \beta_i, \gamma_i\} = \{X_i^{*in} + 1, 2, 0, 0\}$ for all i in $[[1, N_{vars}(X^{in})]]$. Both are compared to find out which one between $f(X_i^{in} = u_{b_i})$ or $f(X_i^{in} = x_i^* + 1)$ yields the minimum value of the objective function. The design variable X_i^{in} is only going to affect the f^{in} part of the objective function f . Hence, it can be assumed that X^{out} (or equivalently f^{out}) is fixed, and only the influence of f^{in} is considered.

Equation A-16: Derivation of the inner loop constrained optimum

$$\begin{aligned}
 0 &\leq x_i^* \leq x_i^* + 1 \leq u_{b_i} \\
 \Rightarrow -x_i^* &\leq 0 \leq 1 \leq u_{b_i} - x_i^* \\
 \Rightarrow 0 &\leq 1 \leq (u_{b_i} - x_i^*)^2 \\
 \Rightarrow f^{in}(x_i^*) &\leq f^{in}(x_i^* + 1) \leq f^{in}(u_{b_i})
 \end{aligned}$$

This proves that $f^{in}(x_i^* + 1) \leq f^{in}(u_{b_i})$ and that $X_i^{in} = x_i^* + 1$ is the global minimum of f^{in} for all i in $\llbracket 1, N_{vars}(X^{in}) \rrbracket$. This can be visualized on Figure A-1, Figure A-2, and Figure A-3.

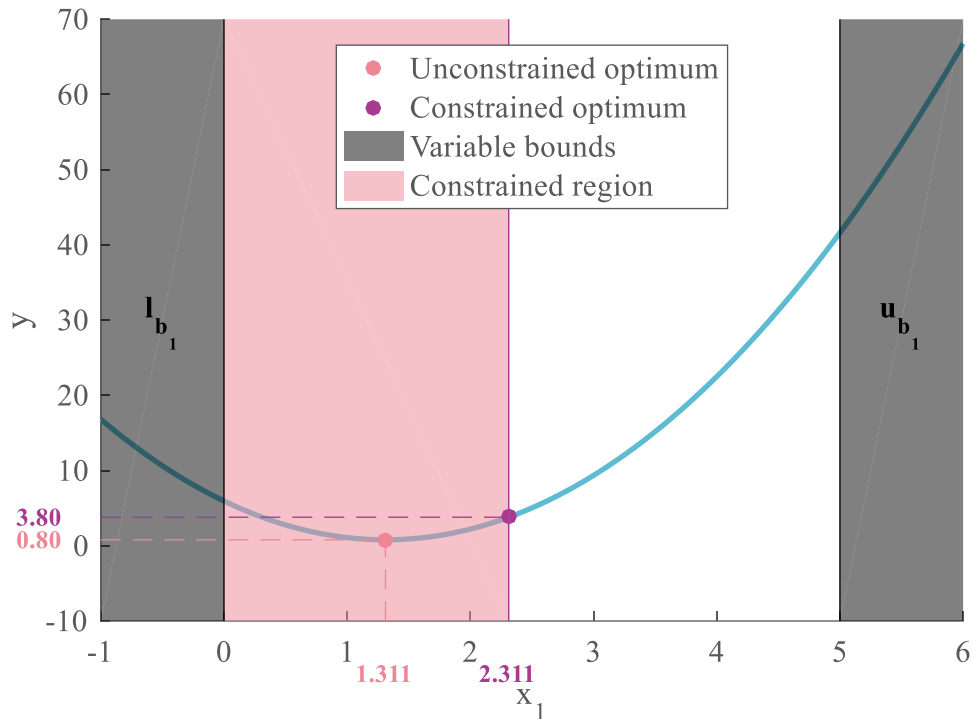


Figure A-1: Constraints on the first design variable

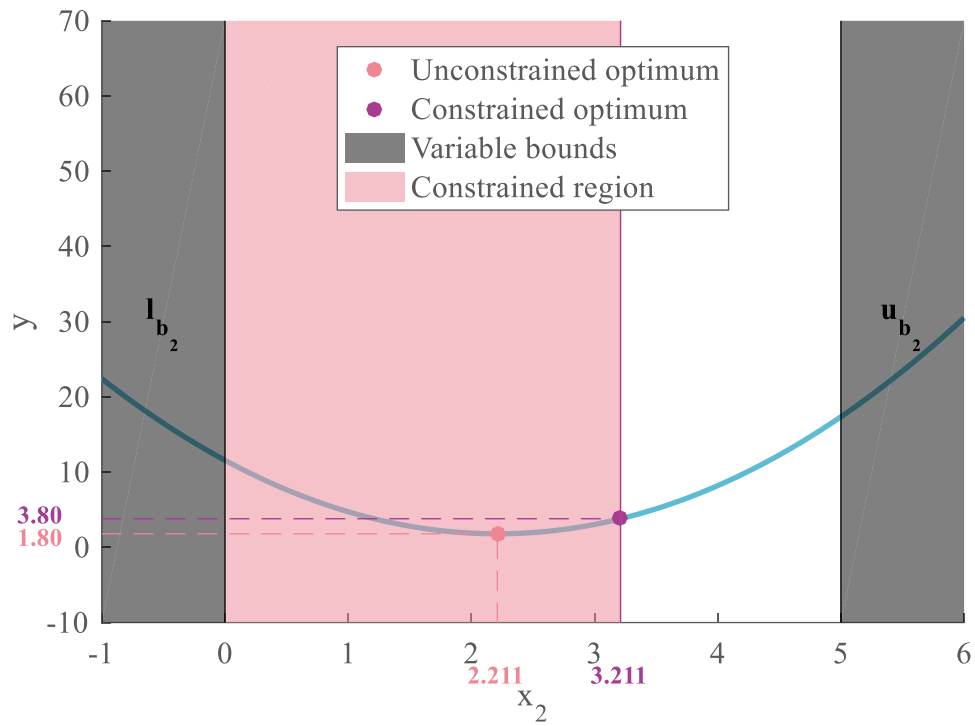


Figure A-2: Constraints on the second design variable

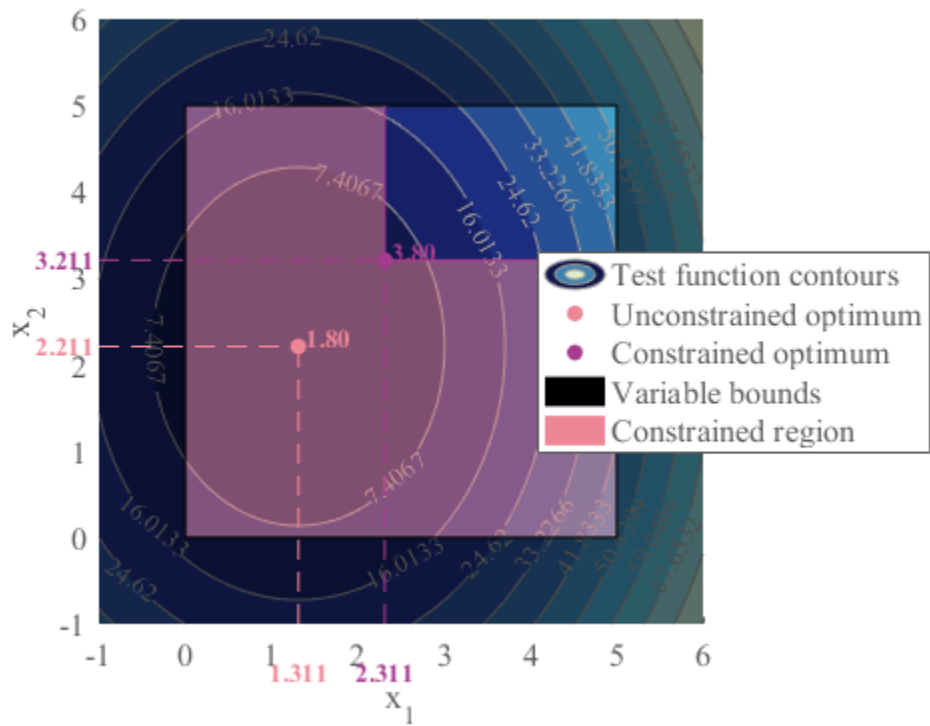


Figure A-3: Constraints on two design variables

Outer loop constraints: the KKT conditions for the outer loop constraints are for all i in

$\llbracket 1, N_{arch} \rrbracket$:

Equation A-17: KKT conditions for the outer loop

$$\left\{ \begin{array}{l} \frac{2(X_i^{out} - i)}{[1 + (X_i^{out} - i)^2]^2} - \delta_i - \epsilon + \zeta - \eta_i + \theta_i = 0 \\ \delta_i(X_i^{*out} + 1 - X_i^{out}) = 0 \\ \epsilon \left(1 - \sum_{i=1}^{N_{arch}} X_i^{out} \right) = 0 \\ \zeta \left(\sum_{i=1}^{N_{arch}} X_i^{out} - N_{max} \right) = 0 \\ \eta_i(-X_i^{out}) = 0 \\ \theta_i(X_i^{out} - N_{max}) = 0 \end{array} \right.$$

Similarly to the analysis used for the inner loop constraints, cases disjunctions have to be made and are detailed in Table A-2.

Table A-2: KKT analysis for outer loop constraints

$\eta_i \neq 0$	$\Rightarrow X_i^{out} = 0$ <p>Impossible since $X_i^{out} \geq X_i^{*out} + 1 = i + 1 \geq 2$</p> $\Rightarrow \eta_i = 0$
$\theta_i \neq 0$	$\Rightarrow X_i^{out} - N_{\max} = 0 \Rightarrow X_i^{out} = N_{\max}$ <p>However, from the constraints:</p> $\sum_{i=1}^{N_{arch}} X_i^{out} \leq N_{\max} \Rightarrow \forall j \neq i, X_j^{out} = 0$ <p>Impossible since $X_i^{out} \geq X_i^{*out} + 1 = i + 1 \geq 2 \forall i \in \llbracket 1, N_{arch} \rrbracket$</p> $\Rightarrow \theta_i = 0$
$\epsilon \neq 0$	$\Rightarrow 1 - \sum_{i=1}^{N_{arch}} X_i^{out} = 0 \Rightarrow \sum_{i=1}^{N_{arch}} X_i^{out} = 1$ <p>However, from the constraints:</p> $\begin{cases} X_i^{out} \in \mathbb{N}, \forall i \in \llbracket 1, N_{arch} \rrbracket \\ X_i^{out} \geq 0, \forall i \in \llbracket 1, N_{arch} \rrbracket \end{cases} \Rightarrow \begin{cases} \exists k, X_k^{out} = 1 \\ \forall i \neq k, X_i^{out} = 0 \end{cases}$ <p>Impossible since $X_i^{out} \geq X_i^{*out} + 1 = i + 1 \geq 2 \forall i \in \llbracket 1, N_{arch} \rrbracket$</p> $\Rightarrow \epsilon = 0$ <p>Note that we consider the case where $N_{arch} \geq 2$.</p>
$\zeta \neq 0$	$\Rightarrow \sum_{i=1}^{N_{arch}} X_i^{out} - N_{\max} = 0 \Rightarrow \sum_{i=1}^{N_{arch}} X_i^{out} = N_{\max}$ <p>This constraint ensures that the total number of vehicles does not exceed a certain limit. For the sake of this analysis and the algorithm verification, this limit is always set high enough so that this constraint is not binding ($N_{\max} \rightarrow +\infty$). Hence, we can assume $\zeta = 0$.</p>
$\delta_i \neq 0$	$\Rightarrow \begin{cases} \frac{2(X_i^{out} - i)}{[1 + (X_i^{out} - i)^2]^2} - \delta_i = 0 \\ X_i^{*out} + 1 - X_i^{out} = 0 \end{cases} \Rightarrow \begin{cases} \delta_i = \frac{2(X_i^{out} - i)}{[1 + (X_i^{out} - i)^2]^2} \\ X_i^{out} = X_i^{*out} + 1 = i + 1 \end{cases}$ $\Rightarrow \begin{cases} \delta_i = \frac{2(i + 1 - i)}{[1 + (i + 1 - i)^2]^2} \\ X_i^{out} = i + 1 \end{cases} \Rightarrow \begin{cases} \delta_i = 1 \\ X_i^{out} = i + 1 \end{cases}$
$\delta_i = 0$	$\Rightarrow \frac{2(X_i^{out} - i)}{[1 + (X_i^{out} - i)^2]^2} = 0 \Rightarrow X_i^{out} = i$ <p>Impossible since $X_i^{out} \geq X_i^{*out} + 1 = i + 1$.</p>

This time, there is only one feasible solution which is $\{X_i^{out}, \delta_i, \epsilon, \zeta, \eta_i, \theta_i\} = \{i + 1, 1, 0, 0, 0, 0\}, \forall i \in \llbracket 1, N_{arch} \rrbracket$.

To conclude the analysis for the constrained verification function, the solution to the constrained optimization problem presented in Equation A-10 is $X = [X^{cout}, X^{cin}]$ with $X_i^{cout} = i + 1, \forall i \in \llbracket 1, N_{arch} \rrbracket$ and $X_i^{cin} = x_i^* + 1, \forall i \in \llbracket 1, N_{vars}(X^{in}) \rrbracket$. These values correspond to the ones announced in section 5.2.2.1.

A.2 Optimization test functions

This appendix details the multivariable optimization test functions used in section 5.1.5 (see page 325) to characterize morphological reduction. The formulae of the functions and their global optimum are given from [236] along with a graphical representation for two variables. It is assumed that the design vector x has dimensionality d .

A.2.1 Ackley

Expression:
$$f(x) = -a \exp\left(-b \sqrt{\frac{1}{d} \sum_{i=1}^d x_i^2}\right) - \exp\left(\frac{1}{d} \sum_{i=1}^d \cos(c x_i)\right) + a + \exp(1)$$

Common values are $\{a, b, c\} = \{20, 0.2, 2\pi\}$

Optimum: $f(x^*) = 0$ at $x^* = [0, 0, \dots, 0]$

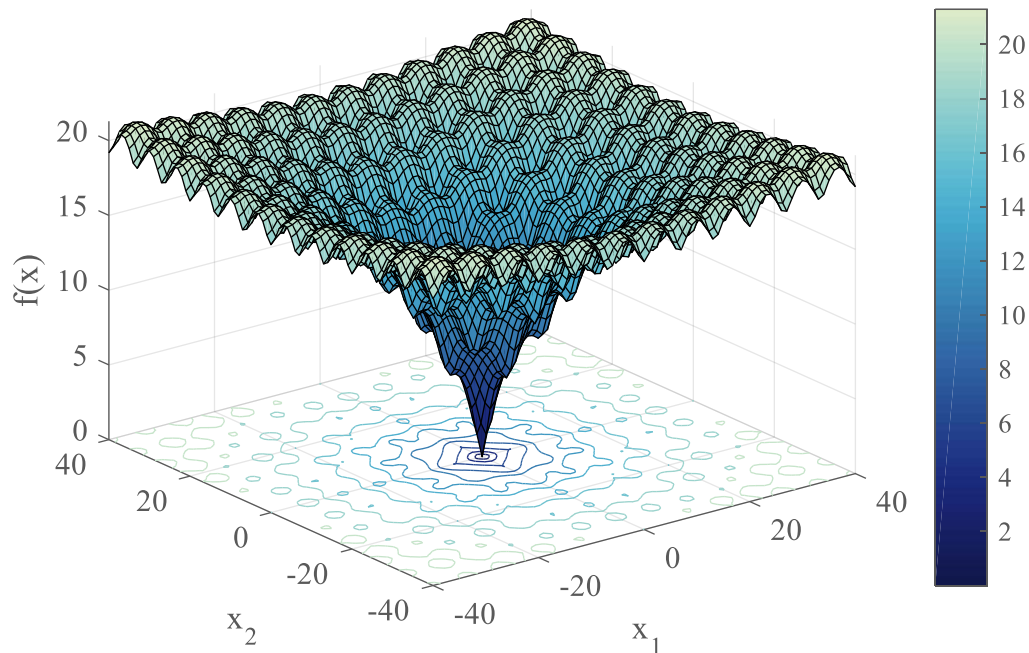


Figure A-4: Ackley function representation

A.2.2 Dixon-Price

Expression: $f(x) = (x_1 - 1)^2 + \sum_{i=2}^d i (2x_i^2 - x_{i-1})^2$

Optimum: $f(x^*) = 0$ at $x_i^* = 2^{-\frac{2^i-2}{2^i}}, \forall i \in \llbracket 1, d \rrbracket$

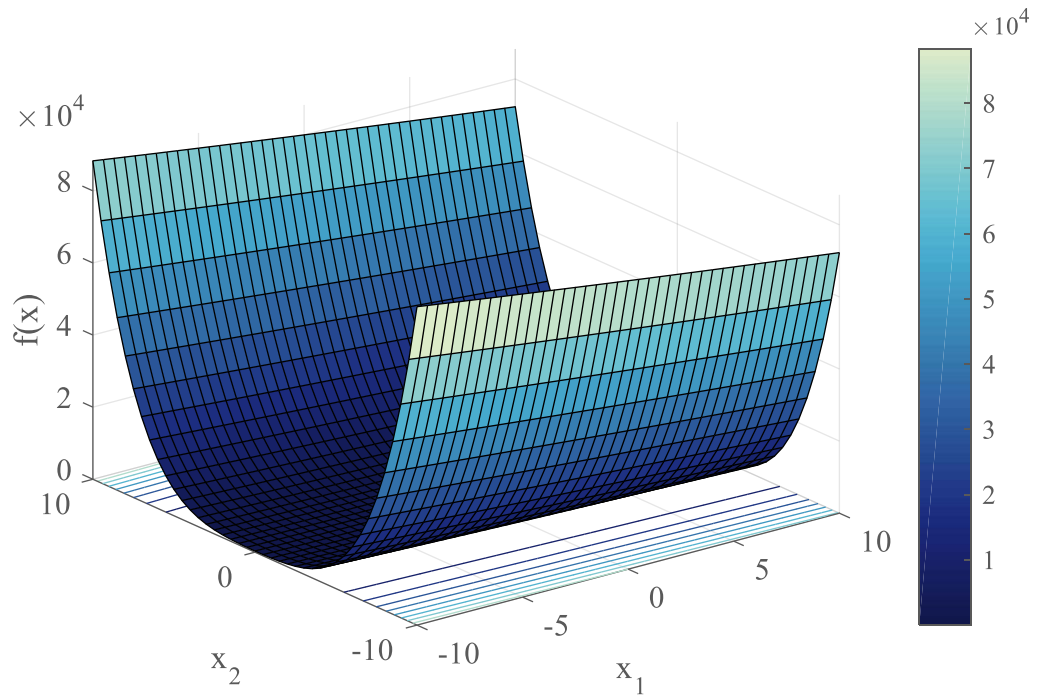


Figure A-5: Dixon-Price function representation

A.2.3 Griewank

Expression: $f(x) = \sum_{i=1}^d \frac{x_i^2}{400} - \prod_{i=1}^d \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$

Optimum: $f(x^*) = 0$ at $x^* = [0,0, \dots,0]$

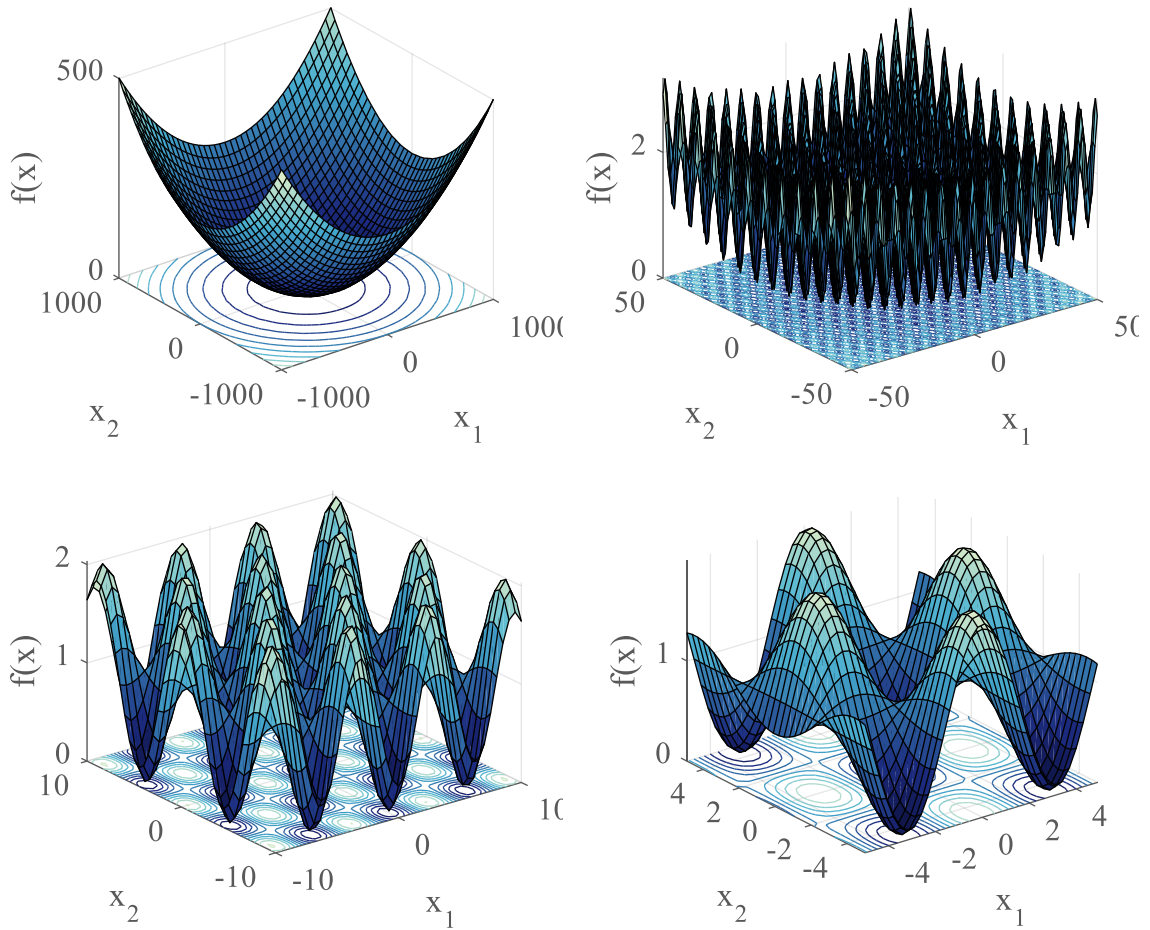


Figure A-6: Griewank function representation over a varied range

A.2.4 Levy

Expression:

$$f(x) = \sin^2(\pi\omega_1) + \sum_{i=1}^{d-1} (\omega_i - 1)^2 [1 + 10 \sin^2(\pi\omega_i + 1)] + (\omega_d - 1)^2 [1 + \sin^2(2\pi\omega_d)]$$

With $\omega_i = 1 + \frac{x_i - 1}{4}, \forall i \in \llbracket 1, d \rrbracket$

Optimum: $f(x^*) = 0$ at $x^* = [1, 1, \dots, 1]$

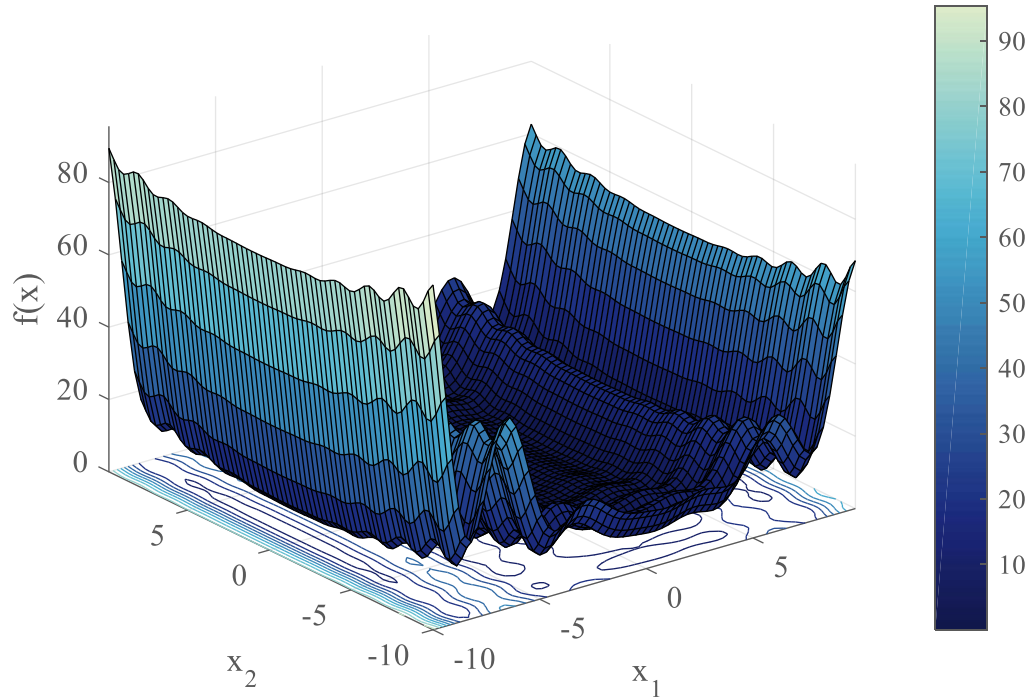


Figure A-7: Levy function representation

A.2.5 Michalewicz

Expression: $f(x) = -\sum_{i=1}^d \sin(x_i) \sin^{2m}\left(\frac{ix_i^2}{\pi}\right)$ with $x_i \in [0, \pi], \forall i \in \llbracket 1, d \rrbracket$

Optimum: presented for the numbers of variables used in section 5.1.5.

Table A-3: Optimum of the Michalewicz test function

Dimensionality d	Global minima value $f(x^*)$	Global minima location x^*
1	-0.8013	2.2029
2	-1.8013	[2.2029, 1.5708]
3	-2.7604	[2.2029, 1.5708, 1.2850]
4	-3.6989	[2.2029, 1.5708, 1.2850, 1.9231]
5	-4.6877	[2.2029, 1.5708, 1.2850, 1.9231, 1.7205]
10	-9.66015	[2.2029, 1.5708, 1.2850, 1.1138, 1.7205, 1.5708, 1.4544, 1.3606, 1.6557, 1.5708]
50	-46.6491	...
100	-88.1784	...

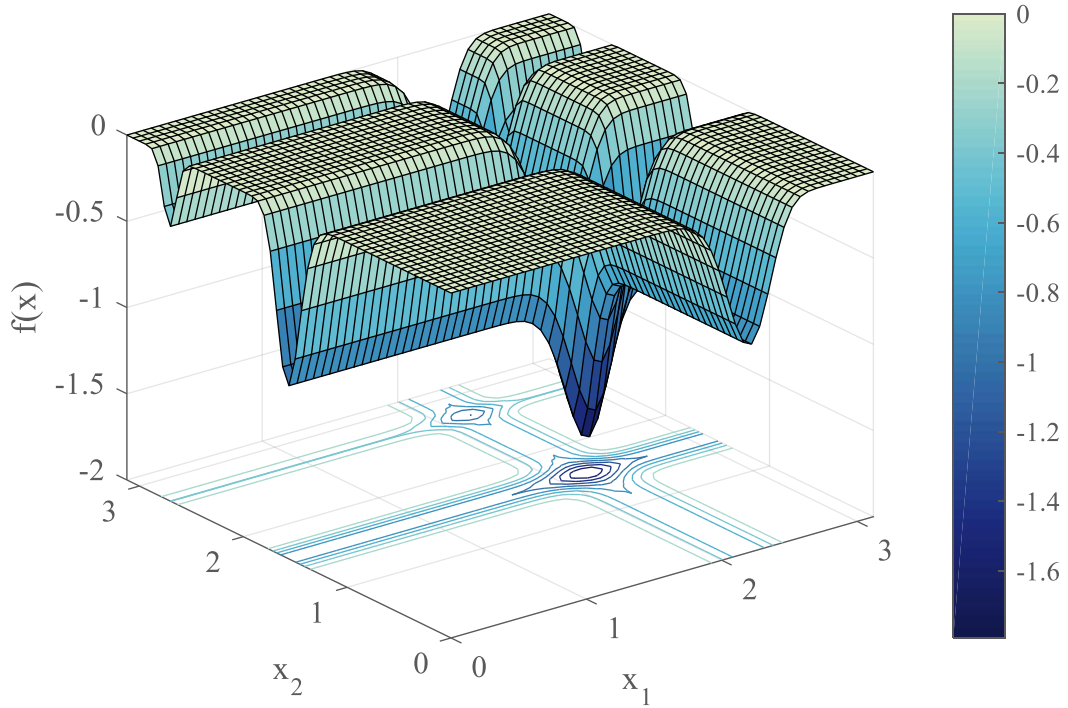


Figure A-8: Michalewicz function representation

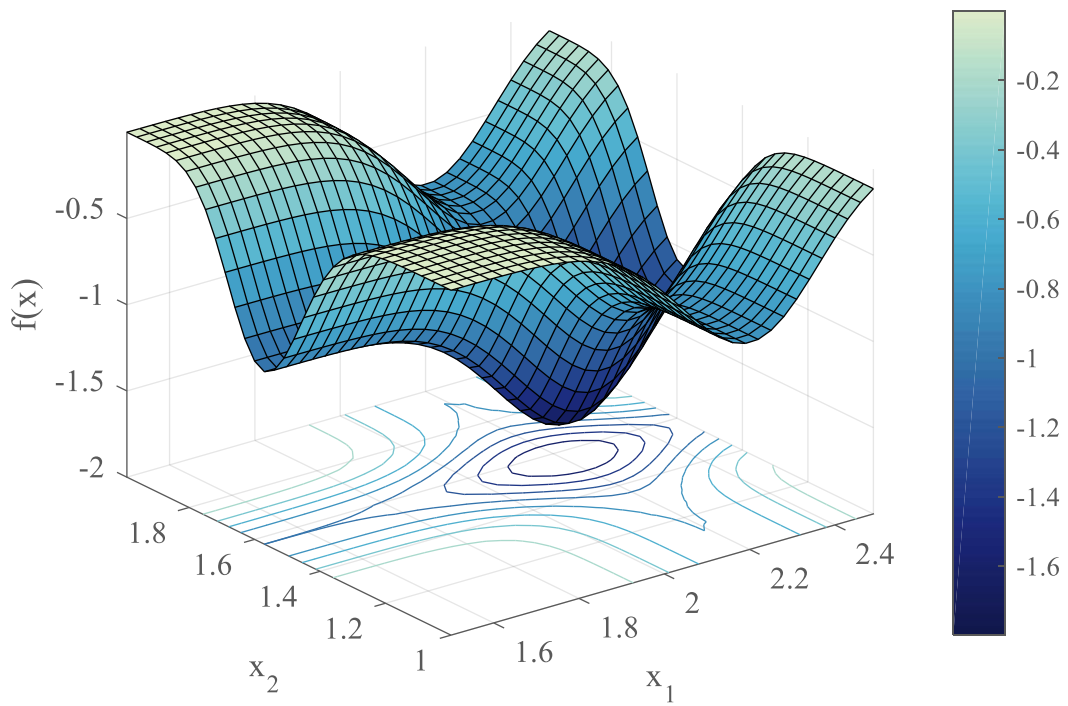


Figure A-9: Local representation of the Michalewicz function

A.2.6 Powell

Expression:

$$f(x) = \sum_{i=1}^{\frac{d}{4}} [(x_{4i-3} + 10x_{4i-2})^2 + 5(x_{4i-1} - x_{4i})^2 + (x_{4i-2} - 2x_{4i-1})^4 + 10(x_{4i-3} - x_{4i})^4]$$

Optimum: $f(x^*) = 0$ at $x^* = [0,0, \dots, 0]$

Given that this function needs at least 4 design variables, no representation is available.

A.2.7 Rastrigin

Expression: $f(x) = 10d + \sum_{i=1}^d [x_i^2 - 10 \cos(2\pi x_i)]$

Optimum: $f(x^*) = 0$ at $x^* = [0,0, \dots, 0]$

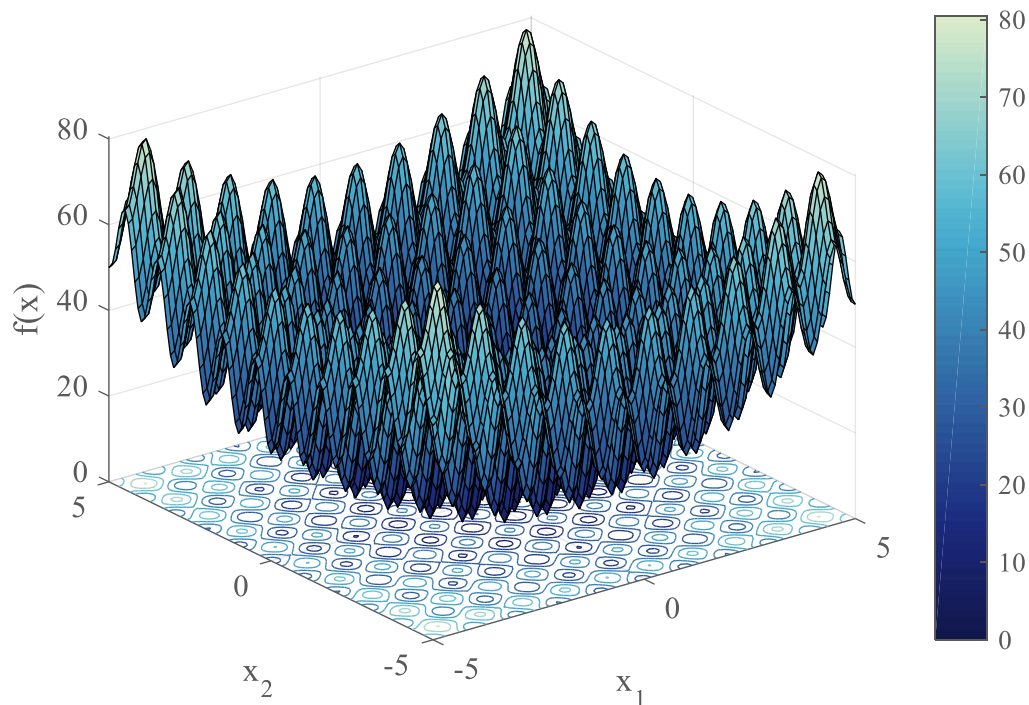


Figure A-10: Rastrigin function representation

A.2.8 Rosenbrock

Expression: $f(x) = \sum_{i=1}^{d-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$

Optimum: $f(x^*) = 0$ at $x^* = [0, 0, \dots, 0]$

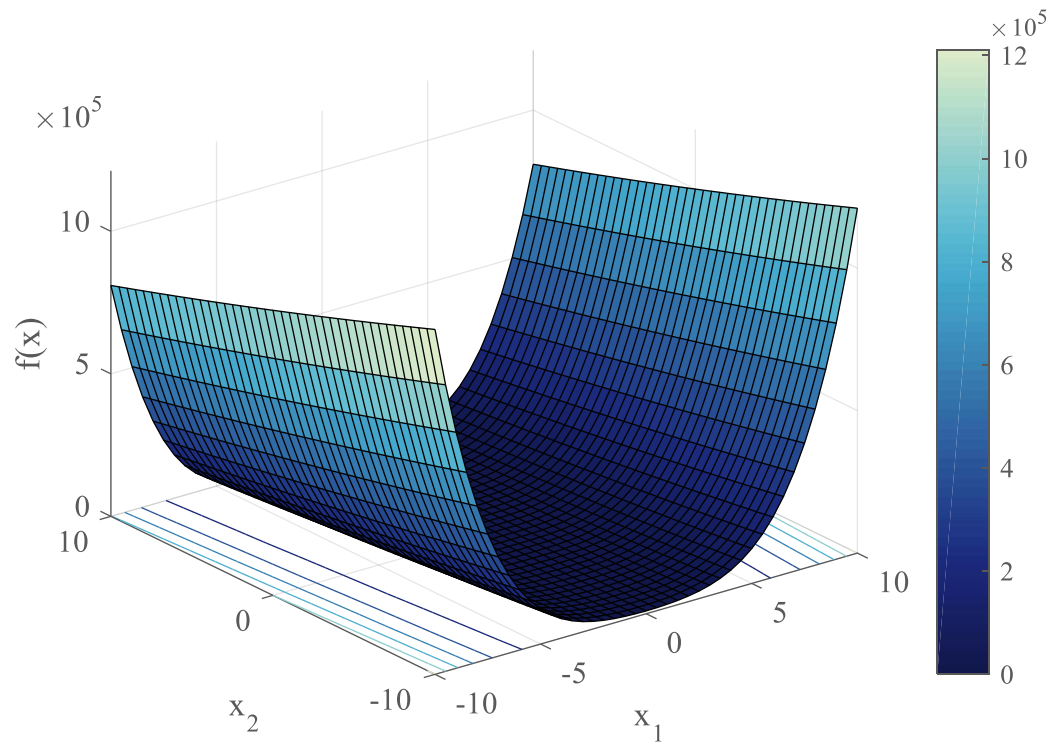


Figure A-11: Rosenbrock function representation

A.2.9 Rotated hyper-ellipsoid

Expression: $f(x) = \sum_{i=1}^d \sum_{j=1}^i x_j^2$

Optimum: $f(x^*) = 0$ at $x^* = [0, 0, \dots, 0]$

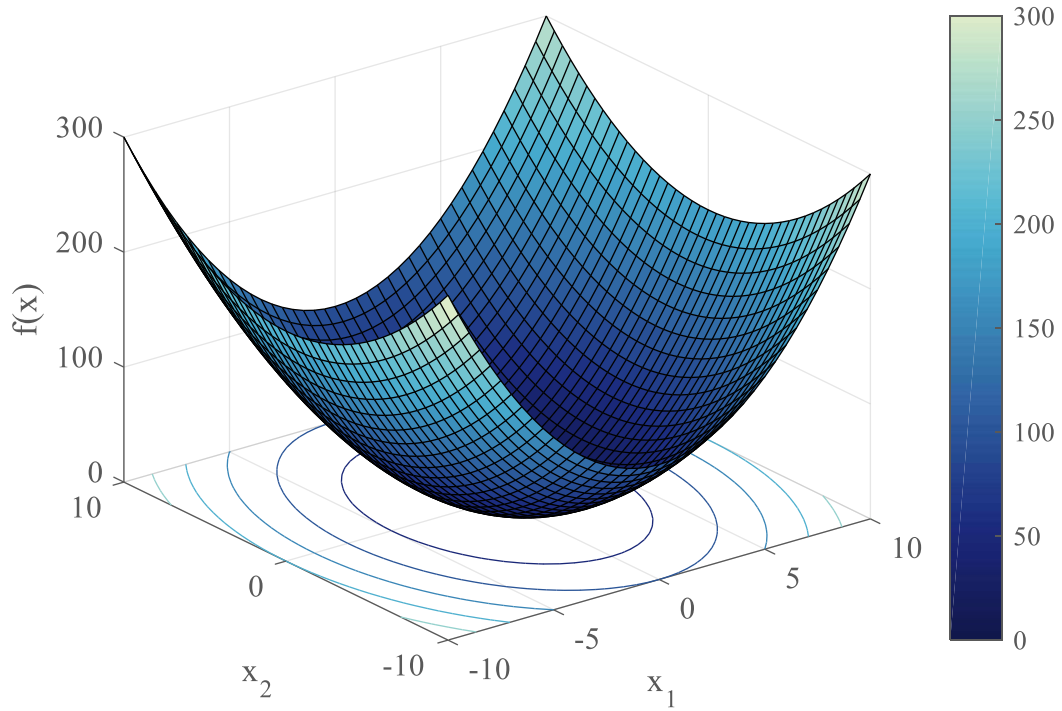


Figure A-12: Rotated Hyper-Ellipsoid function representation

A.2.10 Schwefel

Expression: $f(x) = 418.9829d - \sum_{i=1}^d x_i \sin(\sqrt{|x_i|})$

Optimum: $f(x^*) = 0$ at $x^* = [420.9687, 420.9687, \dots, 420.9687]$

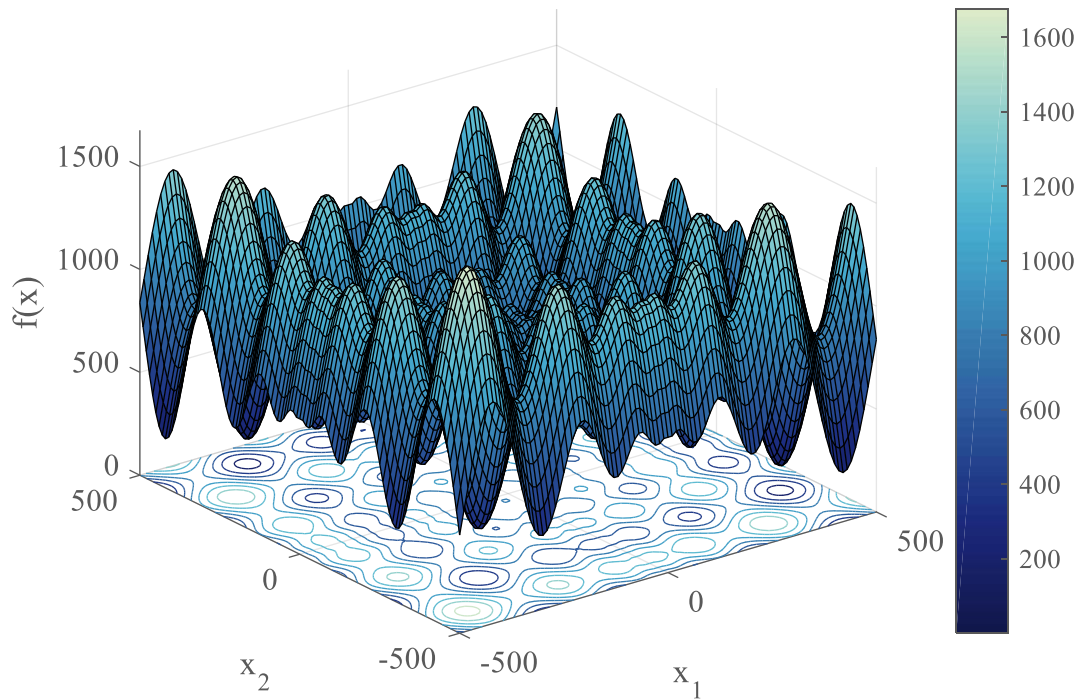


Figure A-13: Schwefel function representation

A.2.11 Sphere

Expression: $f(x) = \sum_{i=1}^d x_i^2$

Optimum: $f(x^*) = 0$ at $x^* = [0,0, \dots, 0]$

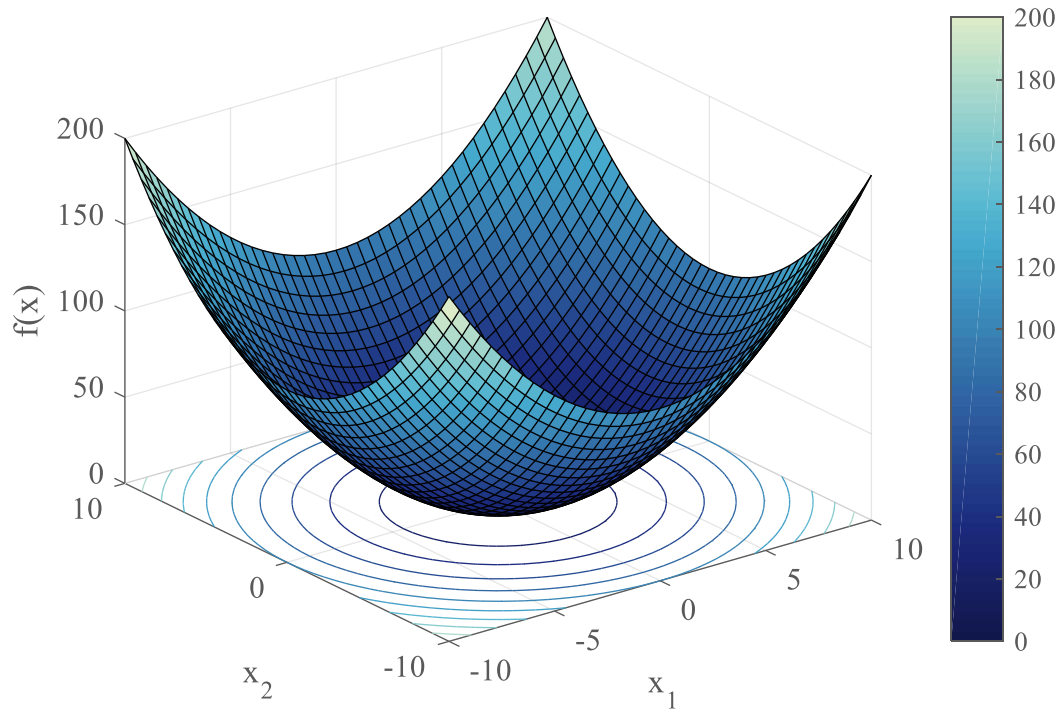


Figure A-14: Sphere function representation

A.2.12 Styblinski-Tang

Expression: $f(x) = \frac{1}{2} \sum_{i=1}^d (x_i^4 - 16x_i^2 + 5x_i)$

Optimum: $f(x^*) = -39.16599$ at $x^* = [-2.903534, -2.903534, \dots, -2.903534]$

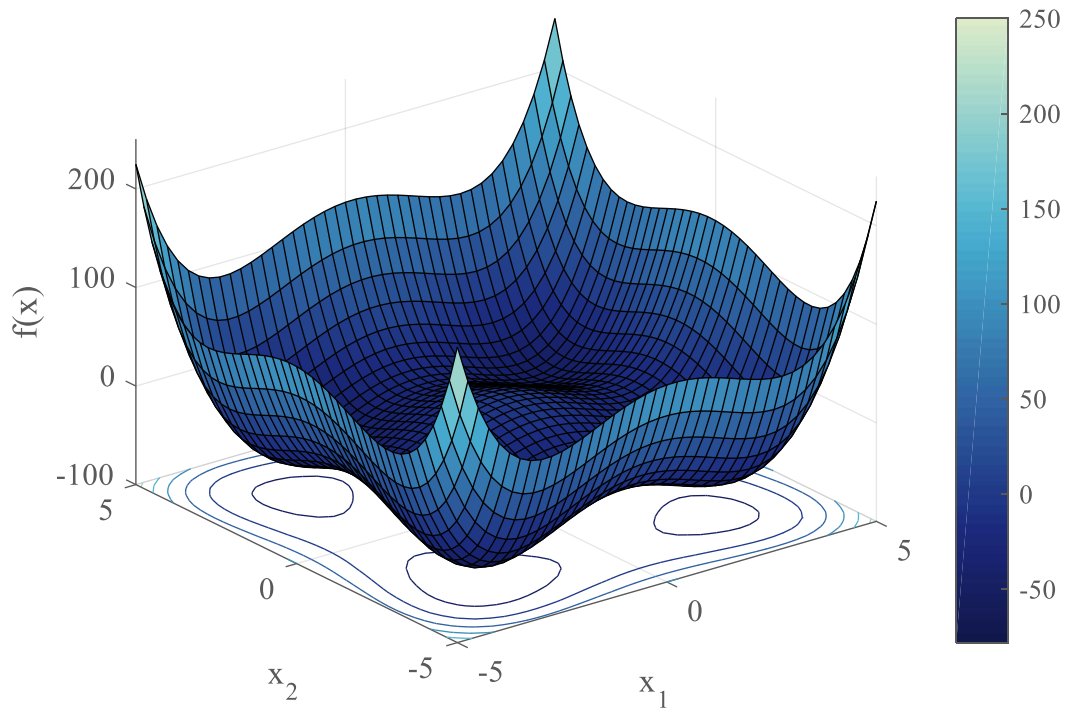


Figure A-15: Styblinski-Tang function representation

A.2.13 Sum squares

Expression: $f(x) = \sum_{i=1}^d ix_i^2$

Optimum: $f(x^*) = 0$ at $x^* = [0,0, \dots, 0]$

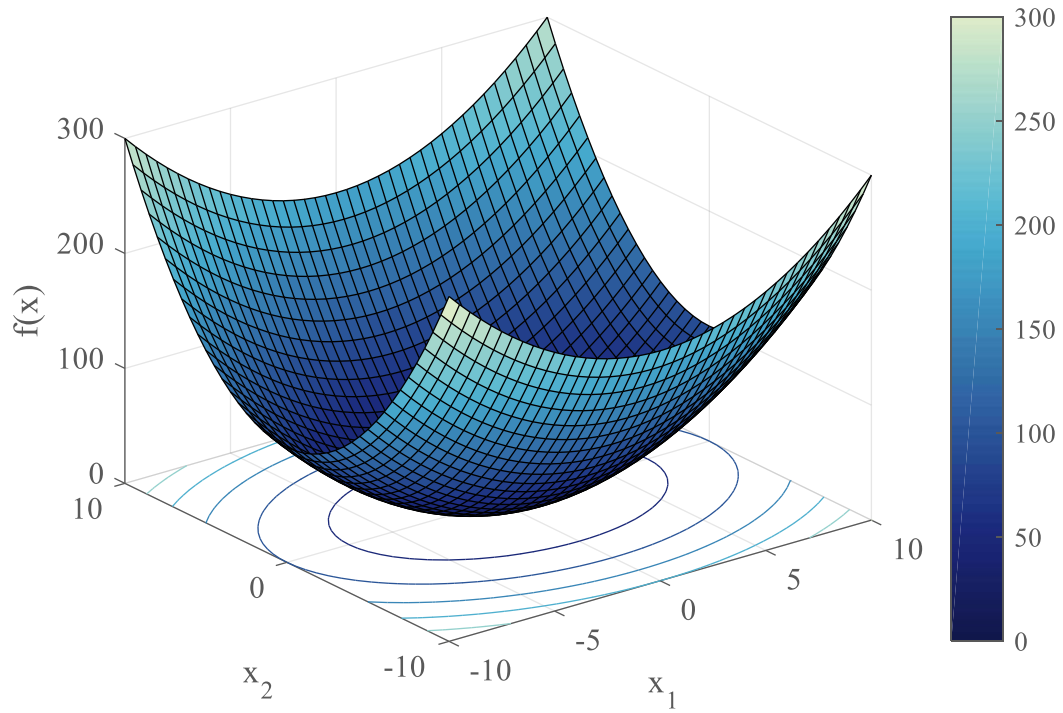


Figure A-16: Sum of squares function representation

A.3 *Barrier certificates formulation*

The example mesoscopic model introduced in section 4.2.4 (see page 274) indirectly models collisions by avoiding them through a barrier certificate method. Using the calculation provided in [224], this appendix provides the detailed formulation of the optimization problem to be solved to compute the collision-safe control commands.

Problem statement: a group of N robots is considered with position states $x_i, \forall i \in \mathcal{M} = \{1, 2, \dots, N\}$. Each robot is controlled through single-integrator dynamics $\dot{x}_i = u_i$ and this velocity command must be bounded by $\|u_i\| \leq \alpha, \forall i \in \mathcal{M}$. The states and velocity commands from all robots are aggregated under notations $x = [x_1^T, x_2^T, \dots, x_N^T]^T$ and $u = [u_1^T, u_2^T, \dots, u_N^T]^T$. The boundary of the testbed is defined by a rectangle $[B_l, B_r, B_b, B_t]$ respectively for left, right, bottom, and top sides. The components of vector x are decomposed such that $x = [x[1], x[2]]^T$.

Theory: the barrier certificate method is based on control barrier functions which are similar to Lyapunov functions in controls theory. In particular, such functions are proven to ensure forward set invariance: if the system starts in a safe set, it remains in it for all time even when subject to small perturbations. In particular, if such a function can be found for the considered multi-robot problem, this will provably ensure that the Robotarium remains collision-free for all time of the experiment.

Inter-robot collisions: first, the distance inequality function between robots i and j ($i \neq j$) is defined as $h_{ij}(x) = \|x_i - x_j\|_2^2 - D_s^2$ where D_s is a safety distance between any given pair of robots. Robots do not collide if $h_{ij}(x) \geq 0, \forall i \neq j$ and [224] shows that h_{ij} is a control barrier function if $\frac{\partial h_{ij}(x)}{\partial x} u \geq -\gamma h_{ij}(x)$ with $\gamma \geq 0$ a margin control parameter. Hence, for the system to remain provably safe, the control input u has to follow the constraints $-2(x_i - x_j)u_i + 2(x_i - x_j)u_j \leq \gamma h_{ij}(x), \forall i \neq j$ which can be formatted as a linear constraint on the aggregate control vector u : $A_{ij}u \leq b_{ij}, \forall i \neq j$ where

Equation A-18: Matricial form of inter-robot collision constraints

$$A_{ij} = \left[0, \dots, \underbrace{-2(x_i - x_j)^T}_{\text{robot } i}, \dots, \underbrace{2(x_i - x_j)^T}_{\text{robot } j}, \dots, 0 \right]$$

$$b_{ij} = \gamma h_{ij}(x)$$

Note that $A_{ij} \in \mathbb{R}^{1 \times 2N}$, $u \in \mathbb{R}^{2N \times 1}$, and $b_{ij} \in \mathbb{R}$.

World boundaries collisions: a safety distance must also be maintained between the robots and the boundaries of the Robotarium arena. The safety inequalities as given by [224] are now $h_{i1}(x) = (B_r - x_i[1])(x_i[1] - B_t) \geq 0$ for the first dimension, and $h_{i2}(x) = (B_t - x_i[2])(x_i[2] - B_b) \geq 0$ for the second. The safety barrier condition ensuring the forward invariance properties is now written $\frac{\partial h_{i1}(x)}{\partial x} u \geq \gamma h_{i1}(x)$ and similarly for h_{i2} which implies:

Equation A-19: World boundaries constraints

$$\begin{cases} (2x_i[1] - B_r - B_l)u_i \leq \gamma h_{i1}(x) \\ (2x_i[2] - B_t - B_b)u_i \leq \gamma h_{i2}(x) \end{cases}, \forall i \in \mathcal{M}$$

Note that the barrier constant γ is the same for all barriers of the problem. Again, this formulation can be formatted as a linear constraint on the controls vector u : $A_i u_i \leq b_i, \forall i \in \mathcal{M}$ with

Equation A-20: Matricial form of world boundaries constraints

$$A_i = \begin{bmatrix} 2x_i[1] - B_r - B_l & 0 \\ 0 & 2x_i[2] - B_t - B_b \end{bmatrix} \in \mathbb{R}^{2 \times 2}$$
$$b_i = \begin{bmatrix} \gamma h_{i1}(x) \\ \gamma h_{i2}(x) \end{bmatrix} \in \mathbb{R}^{2 \times 1}$$

Minimally invasive control: thanks to the control barrier functions defined in the previous paragraphs, a forward invariant safe set can be constructed with the safety barrier certificates $\{u \in \mathbb{R}^{2N \times 1} | A_{ij}u \leq b_{ij}, A_i u_i \leq b_i, \forall i \neq j\}$ which defined a convex polytope within which the control commands will guarantee a collision-free behavior of the Robotarium for all time. The idea is then to choose a control within this set which is optimal in some way. One possible solution is to insure that the new optimal control u^* is not too different from the control \hat{u} imposed by the user or the control scheme. Indeed, it is essential that, while collisions are avoided, the mission is still performed as initially planned.

Equation A-21: Quadratic program based controller

$$u^* = \underset{u \in \mathbb{R}^{2N \times 1}}{\operatorname{argmin}} \left(\sum_{i \in \mathcal{M}} \|u_i - \hat{u}_i\|^2 \right)$$
$$s. t. \begin{cases} A_{ij}u \leq b_{ij} & \forall i \neq j \\ A_i u_i \leq b_i & \forall i \in \mathcal{M} \\ \|u_i\|_\infty \leq \alpha & \forall i \in \mathcal{M} \end{cases}$$

This is the principle of the minimally invasive method which consists in finding the certificate barrier control which differs the least (in terms of the 2-norm) from the initial control (see Equation A-21). Given that this optimization problem is quadratic and defined over a polytope, quadratic programming optimization can be used. If the safety barrier certificates are not violated, the initial control \hat{u} is chosen and there is no modification from the initial single-integrator dynamics. In the case when the controls of the user would cause collisions (or robots being too close to each other with respect to the safety distance), the “closest” safe control command is chosen.

APPENDIX B

MATLAB CODE

This appendix regroups the code implemented using Matlab for each domain of the research questions: modeling, optimization, and exploration. It also includes the different scripts which were used to automate files generation, simulation, and results analysis.

B.1 Modeling

The files used to carry out the experiments for each type of modeling technique and then compare the results.

B.1.1 Macroscopic

rendezvousMacroscopic.m

The main model which outputs a consensus time, position, and an execution time as a function of a swarm configuration (number of robots and initial velocity), and initial poses.

```
function [t,x,time] = rendezvousMacroscopic(v,N,initialPositions)
tic
% Final position
x = 1/N * sum(initialPositions(1:2,:),2);
% Time to reach consensus
t = max(sqrt(sum((initialPositions(1:2,:) - repmat(x,1,N)).^2,1)))/v;
time = toc;
end
```

doe_macroscopic.m

A script executing and collecting the results of the 40 experiments presented in section 4.2.6 (page 281).

```
% Prepare workspace
clc
close all
clear
```

```

%% Design space
Nvec = 2:5;
Vvec = .01:0.01:0.1;

[N,V] = meshgrid(Nvec,Vvec);

%% Analysis
T_macro = zeros(size(N));
X_macro = cell(size(N));
t_macro = zeros(size(N));
n1 = size(N,1);
n2 = size(N,2);
for i = 1:n1
    for j = 1:n2
        fprintf('%d/%d (%.2f%%)\n', (i-1)*n2 + j, n1*n2, 100*((i-1)*n2 +
j)/(n1*n2))

        % Load initial positions
load(sprintf('../initialPoses/init_poses_%d_%.2f.mat',N(i,j),V(i,j)))

        % Perform model analysis
[t,x,time] = rendezvousMacroscopic(V(i,j),N(i,j),x0);

        % Store results
T_macro(i,j) = t;
X_macro{i,j} = x;
t_macro(i,j) = time;
    end
end

save('macroscopic.mat')

```

B.1.2 Mesoscopic

rendezvousMesoscopic.m

The main model which outputs a consensus time, position, and an execution time as a function of a swarm configuration (number of robots and initial velocity), and initial poses. This model uses a modified version of the Robotarium simulator available online. Indeed, the ability to control the number of robots generated is essential here. The main Robotarium constructor was hence modified to use as an input a given number of Gritbots (see Robotarium files here below).


```

% Experiment
% 1 - Initializes N robots at initial_conditions
% 2 - Performs static consensus with a maximum linear speed saturated
at v
%
% Jean-Guillaume Durand
% jdurand7@gatech.edu
% 2016
function [t,x,time] = rendezvousMesoscopic(v,N,initialPositions)
% Start timer
tic

% Format initial conditions correctly
initial_conditions = initialPositions;

%% 1 - Initialize N robots at initial conditions
% Get Robotarium object used to communicate with the robots/simulator
rb = RobotariumBuilder(N);

% Get the number of available agents from the Robotarium. We don't
need a
% specific value for this algorithm
N_available = rb.get_available_agents();

% If not enough robots for experiment, stop
if N_available < N, return, end

% Set the number of agents and whether we would like to save data.
Then,
% build the Robotarium simulator object!
r =
rb.set_number_of_agents(N).set_save_data(false).build(initial_conditions);

% Initialize x so that we don't run into problems later. This isn't
always
% necessary
x = r.get_poses();
r.step();

% Set some parameters for use with the barrier certificates. We don't
want
% our agents to collide
safety = 0.06;
lambda = 0.03;

% Create a barrier certificate for use with the above parameters
uncycle_barrier_certificate =
create_uni_barrier_certificate('SafetyRadius', safety, ...
    'ProjectionDistance', lambda);

% Create parking controller
args = {'PositionError', 0.01, 'RotationError', 0.1};
init_checker = create_is_initialized(args{:});
automatic_parker = create_automatic_parking_controller(args{:});

```

```

while(~init_checker(x, initial_conditions))
    % Compute velocities
    x = r.get_poses();
    dxu = automatic_parker(x, initial_conditions);
    dxu = unicycle_barrier_certificate(dxu, x);
    % Update
    r.set_velocities(1:N, dxu);
    r.step();
end

%% 2 - Perform static consensus with a maximum linear speed saturated
at v
% Experiment constants
% Generate a cyclic graph Laplacian from our handy utilities. For this
% algorithm, any connected graph will yield consensus
L = cycleGL(N);

% Grab tools we need to convert from single-integrator to unicycle
dynamics
% Gain for the diffeomorphism transformation between single-integrator
and
% unicycle dynamics
[si_to_uni_dyn, uni_to_si_states] =
create_si_to_uni_mapping('ProjectionDistance', lambda);

si_barrier_cert = create_si_barrier_certificate('SafetyRadius',
safety);

% Select the number of iterations for the experiment. This value is
% arbitrary
iterations = 5000; % Maximum time at around 3 minutes

% Initialize velocity vector for agents. Each agent expects a 2 x 1
% velocity vector containing the linear and angular velocity,
respectively.
dxi = zeros(2, N);

xOld = -100*ones(3,N);

%Iterate for the previously specified number of iterations
for it = 1:iterations
    % Retrieve the most recent poses from the Robotarium. The time
delay is
    % approximately 0.033 seconds
    x = r.get_poses();

    % Test for stopping condition
    dx = x(1,:) - xOld(1,:);
    dy = x(2,:) - xOld(2,:);
    if mean(sqrt(dx.^2 + dy.^2)) < 1e-5
        break;
    end
    xOld = x;
end

```

```

% Convert to SI states
xi = uni_to_si_states(x);

% Algorithm
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i = 1:N
    % Initialize velocity to zero for each agent. This allows us
to sum
    %over agent i's neighbors
    dxi(:, i) = [0 ; 0];

    % Get the topological neighbors of agent i based on the graph
    %Laplacian L
    neighbors = topological_neighbors(L, i);

    % Iterate through agent i's neighbors
    for j = neighbors

        % For each neighbor, calculate appropriate consensus term
and
        %add it to the total velocity
        dxi(:, i) = dxi(:, i) + (xi(:, j) - xi(:, i));
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Utilize barrier certificates
dxi = si_barrier_cert(dxi, xi);

% Transform the single-integrator to unicycle dynamics using the
the
% transformation we created earlier
dxu = si_to_uni_dyn(dxi, x);

% Impose velocity v on agents
linear = dxu(1,:);
linear(linear > v) = v;
linear(linear < -v) = -v;
dxu(1,:) = linear;

% Set velocities of agents 1,...,N
r.set_velocities(1:N, dxu);

% Send the previously set velocities to the agents. This function
must be called!
r.step();
end

% Though we didn't save any data, we still should call
r.call_at_scripts_end() after our
% experiment is over!
r.call_at_scripts_end();

% Compute results

```

```

t = it*r.time_step; % Time to reach consensus
x = mean(r.poses(1:2,:),2); % Final consensus position
time = toc; % Time for model to run
end

```

doe_mesoscopic.m

```

% Prepare workspace
clc
close all
clear

% Initialize Robotarium
cd('robotarium-matlab-simulator-master')
run('init.m')
cd('.') % Go back to initial folder

%% Design space
Nvec = 2:5;
Vvec = .01:0.01:0.1;

[N,V] = meshgrid(Nvec,Vvec);

%% Analysis
T_meso = zeros(size(N));
X_meso = cell(size(N));
t_meso = zeros(size(N));
n1 = size(N,1);
n2 = size(N,2);
for i = 1:n1
    for j = 1:n2
        fprintf('%d/%d (%.2f%%)\n', (i-1)*n2 + j, n1*n2, 100*((i-1)*n2 +
j)/(n1*n2))

        % Load initial positions
load(sprintf('../initialPoses/init_poses_%d_%.2f.mat',N(i,j),V(i,j)))

        % Perform model analysis
[t,x,time] = rendezvousMesoscopic(V(i,j),N(i,j),x0);

        % Store results
T_meso(i,j) = t;
X_meso{i,j} = x;
t_meso(i,j) = time;
    end
end

save('mesoscopic.mat')

```

The following files are the modified Robotarium files.

ARobotarium.m

```
classdef ARobotarium < handle
    %APIAbstract This is an interface for the Robotarium class that
    %ensures the simulator and the robots match up properly. You
    should
    %definitely NOT MODIFY this file. Also, don't submit this file
    with
    %your algorithm.

    properties (GetAccess = protected, SetAccess = protected)
        robot_handle
        robot_body

        % Stuff for saving data
        file_path
        current_file_size
        current_saved_iterations
        % Path to mat file to keep this in memory
        mat_file_path
    end

    properties (GetAccess = public, SetAccess = protected)
        % Time step for the Robotarium
        time_step = 0.033
        maxLinearVelocity = 0.1
        maxAngularVelocity = 2*pi
        robot_diameter = 0.08
        number_of_agents
        velocities
        poses

        %Saving parameters
        save_data

        % Figure handle for simulator
        figure_handle

        % Arena parameters
        boundaries = [-0.6, 0.6, -0.35, 0.35];
        boundary_points = {[-0.6, 0.6, 0.6, -0.6], [-0.35, -0.35, 0.35,
0.35]};
    end

    methods (Abstract)

        %Try this one out...
        % We can use this to finish saving / clean up after MQTT
        call_at_scripts_end(this)

        % Getters
        % Get poses must be implemented independently
        get_poses(this)
    end
end
```

```

        %Update functions
        step(this);
    end

    methods
        function this = ARobotarium(number_of_agents, save_data,
initial_poses)
            this.number_of_agents = number_of_agents;
            this.save_data = save_data;

            this.velocities = zeros(2, number_of_agents);
            this.poses = initial_poses;

            % If save data, set up the file saving variables
            if(save_data)
                date = datetime('now');
                this.file_path = 'robotarium_data';
                this.file_path = strcat(this.file_path, '_',
num2str(date.Month), '_', num2str(date.Day), '_', ...
num2str(date.Year), '_', num2str(date.Hour), '_', ...
num2str(date.Minute), '_', num2str(round(date.Second)),
'.mat');

                this.current_file_size = 100;
                this.current_saved_iterations = 1;

                robotarium_data = zeros(5*number_of_agents,
this.current_file_size);
                save(this.file_path, 'robotarium_data', '-v7.3')

                this.mat_file_path = matfile(this.file_path,
'Writable', true);
            end
        end

        function agents = get_number_of_agents(this)
            agents = this.number_of_agents;
        end

        function this =set_velocities(this, ids, vs)
            N = size(vs, 2);

            assert(N<=this.number_of_agents, 'Column size of vs (%i)
must be <= to number of agents (%i)', ...
                N, this.number_of_agents);

            % Threshold velocities
            for i = 1:N
                if(abs(vs(1, i)) > this.maxLinearVelocity)
                    vs(1, i) = this.maxLinearVelocity*sign(vs(1,i));
                end

                if(abs(vs(2, i)) > this.maxAngularVelocity)
                    vs(2, i) = this.maxAngularVelocity*sign(vs(2, i));
                end
            end
        end
    end
end

```

```

        end

        this.velocities(:, ids) = vs;
    end

    function iters = time2iters(this, time)
        iters = time / this.time_step;
    end
end

methods (Access = protected)

% Initializes visualization of GRITSBots
function initialize_visualization(this)
    % Initialize variables
    numRobots = this.number_of_agents;
    offset = 0.05;

    % Scale factor (max. value of single Gaussian)
    scaleFactor = 0.5;
    figPhi = figure;
    this.figure_handle = figPhi;

    % Plot Robotarium boundaries
    patch('XData', this.boundary_points{1}, 'YData',
this.boundary_points{2}, ...
        'FaceColor', 'none', ...
        'LineWidth', 3, ...
        'EdgeColor', [0, 0.74, 0.95]);

    %plot(im)
    set(figPhi, 'color', 'white', 'menubar', 'none');

    % Set axis
    robotPlaneAxes = gca;

    % Limit view to xMin/xMax/yMin/yMax
    axis(robotPlaneAxes, [this.boundaries(1) -
offset, this.boundaries(2)+offset, this.boundaries(3) -
offset, this.boundaries(4)+offset])
    caxis([0, 1.5*scaleFactor])
    set(robotPlaneAxes, 'PlotBoxAspectRatio', [1 1
1], 'DataAspectRatio', [1 1 1])

    % Store axes
    axis(robotPlaneAxes, 'off')

    set(robotPlaneAxes, 'position', [0 0 1
1], 'units', 'normalized', 'YDir', 'normal')

    hold on % "This ride's about to get bumpy!"

    % Let's jump through hoops to make the robot diameter look
to data scale

```

```

curUnits = get(robotPlaneAxes, 'Units');
set(robotPlaneAxes, 'Units', 'Points');
set(robotPlaneAxes, 'Units', curUnits);

xlim([-0.65, 0.65]); ylim([-0.35, 0.35]); % static limits

% Static legend
%set(gca, 'LegendColorbarListeners', []);
setappdata(gca, 'LegendColorbarManualSpace', 1);
setappdata(gca, 'LegendColorbarReclaimSpace', 1);

assert(numRobots <= 100, 'Number of robots (%i) must be <=
100', numRobots);

this.robot_handle = cell(1, numRobots);
%load('patches.mat');
patches = gritsbot_patch(100);
num_patches = numel(patches);
chosen_patches = randsample(1:num_patches, numRobots);
patch_data = patches(chosen_patches);
for ii = 1:numRobots
    data = patch_data{ii};
    this.robot_body = data.robot_body;
    x = this.poses(1, ii);
    y = this.poses(2, ii);
    th = this.poses(3, ii);
    poseTransformationMatrix = [...
        cos(th) -sin(th) x;
        sin(th)  cos(th) y;
        0 0 1];
    robot_bodyTransformed =
data.robot_body*poseTransformationMatrix';
    this.robot_handle{ii} = patch(...
        'Vertices', robot_bodyTransformed, ...
        'Faces', data.robot_face, ...
        'FaceColor', 'flat', ...
        'FaceVertexCData', data.robot_color, ...
        'EdgeColor', 'none');
end
end

function draw_robots(this)
    for ii = 1:this.number_of_agents
        x = this.poses(1, ii);
        y = this.poses(2, ii);
        th = this.poses(3, ii);
        poseTransformationMatrix = [...
            cos(th) -sin(th) x;
            sin(th)  cos(th) y;
            0 0 0 1 1];
        robotBodyTransformed =
this.robot_body*poseTransformationMatrix';
        set(this.robot_handle{ii}, 'Vertices',
robotBodyTransformed);
    end
end

```



```

        if(this.number_of_agents <= 6)
            drawnow
        else
            drawnow limitrate
        end
    end

    function save(this)
        this.mat_file_path.robotarium_data(:,
this.current_saved_iterations) = ...
            reshape([this.poses ; this.velocities], [], 1);

%           % Use array list expansion criterion to amortize file
%           % expansions
%           if(this.current_saved_iterations >
(this.current_file_size / 2))
%               new_robotarium_data = zeros(5*this.number_of_agents,
this.current_file_size * 2);
%               new_robotarium_data(:,
1:this.current_saved_iterations) = ...
%                   this.mat_file_path.robotarium_data(:,
1:this.current_saved_iterations);
%
%           % Set file to new data
%           this.mat_file_path.robotarium_data =
new_robotarium_data;
%           end

        this.current_saved_iterations =
this.current_saved_iterations + 1;
    end
end
end
end

```

ARobotariumBuilder.m

```

classdef ARobotariumBuilder < handle
    %ARobotariumBuilder This is an abstract class for the
RobotariumBuilder class
    %that models the manner in which a Robotarium object is created
    % This file should never be modified.  Otherwise, your code will
not
    % execute properly on the Robotarium

    properties (GetAccess = public, SetAccess = protected)
        available_agents
        number_of_agents
        save_data = true
    end

    methods (Abstract)
        % Builds the Robotarium object.  Definitely backend/sim
dependent.
    end
end

```

```

        get_available_agents(this);
        build(this);
    end

    methods
        function this = set_number_of_agents(this, number_of_agents)

            assert(number_of_agents > 0, 'The provided number of agents
(%i) must be > 0', number_of_agents);

            this.number_of_agents = number_of_agents;
        end

        function this = set_save_data(this, save_data)

            assert(save_data >= 0 || save_data < 0, 'Save data must
evaluate to true or false in a boolean expression');

            this.save_data = save_data;
        end
    end
end

```

RobotariumBuilder.m

```

%% RobotariumBuilder
% This class handles the creation of the Robotarium object. In
particular,
% it controls and sets the parameters for your simulation/experiment.
%% Function Summary
% * get_available_agents():  $\emptyset$  to  $\mathbf{Z}^{\{+\}}$  returns the
number of available agents
% (random for each instantiation)
% * set_number_of_agents():  $\mathbf{Z}^{\{+\}}$  to RobotariumBuilder$ sets
the number of
% agents, returning the RobotariumBuilder object
% * set_save_data():  $\{false, true\}$  to RobotariumBuilder$ sets
whether
% to save data for this experiment.
% * build():  $\emptyset$  to Robotarium$ builds a Robotarium object with
the
% specified parameters
%% Example Usage
% % Example showing potential usage of the RobotariumBuilder object.
% % Note that get_available_agents() returns the number of available
% % agents, which is random. If you need a particular number of
agents,
% % this should be specified in the experiment descriptor when you
% % eventually submit your experiment to the Robotarium.
% % Or you can design your experiment to handle any number of agents.
%
% % set_save_data() controls whether the Robotarium records
% % your simulation/experimental data.

```

```

%
% robo_builder = RobotariumBuilder()
% N = robo_builder.get_available_agents()
% robo_obj =
% robo_builder.set_number_of_agents(N).set_save_data(true).build()

classdef RobotariumBuilder < ARobotariumBuilder
    %ROBOTARIUMBUILDER This class handles creation of the Robotarium
    object
    %that communicates with the GRITsbots.
    % This class is really just a helper to assist with creating the
    % Robotarium object. In particular, this object allows you to set
    % properties for your simulation and eventual experiment. Right
    now,
    % these properties are: number of agents and whether to save data.

    % THIS CLASS SHOULD NEVER BE MODIFIED

    % Gets properties from abstract class as well.
    properties
        boundaries = [-0.6, 0.6, -0.35, 0.35];
        robot_diameter = 0.08
    end

    methods

        function this = RobotariumBuilder(N)
            this.available_agents = N;%randi(14) + 1;
            this.number_of_agents = -1;
        end

        function number_of_agents = get_available_agents(this)
            number_of_agents = this.available_agents;
        end

        function robotarium_obj = build(this,initial_poses)

            assert(this.number_of_agents > 0, 'You must set the number
of agents for this experiment');

            arena_width = this.boundaries(2) - this.boundaries(1);
            arena_height = this.boundaries(4) - this.boundaries(3);

            numX = floor(arena_width / this.robot_diameter);
            numY = floor(arena_height / this.robot_diameter);
            values = randperm(numX * numY, this.number_of_agents);

            initial_poses = zeros(3, this.number_of_agents);

            for i = 1:this.number_of_agents
                [x, y] = ind2sub([numX numY], values(i));
                x = x*this.robot_diameter - (arena_width/2);
                y = y*this.robot_diameter - (arena_height/2);
                initial_poses(1:2, i) = [x ; y];
            end
        end
    end
end

```

```

%
%           initial_poses(3, :) = rand(1,
this.number_of_agents)*2*pi;
           robotarium_obj = Robotarium(this.number_of_agents,
this.save_data, initial_poses);
       end
   end
end

```

Robotarium.m

```

%% Robotarium
% A class that models your communications with the GRITSBots!
% This class handles retrieving the poses of agents, setting their
% velocities, iterating the simulation, and saving data.
%% Method Description
% * get_poses():  $\emptyset$  to  $\mathbf{R}^{3 \times N}$  retrieves the
% poses of the agents in a 3 x N vector, where each column contains the
% pose of an agent.
% * set_velocities():  $\mathbf{R}^{2 \times N}$  to Robotarium sets the
% velocities of each agent using a 2 x N vector. Each column
represents
% the linear and angular velocity of an agent.
% * step():  $\emptyset$  to  $\emptyset$  iterates the simulation, updating
the
% state of each agent. This function should be called for each
"iteration"
% of your experiment. Additionally, it should only be called once per
call
% of get_poses().

classdef Robotarium < ARobotarium
    %Robotarium This is the Robotarium simulator object that represents
    %your communications with the GRITSBots.
    % This class handles retrieving the poses of agents, setting
their
    % velocities, iterating the simulation, and saving data.

    % THIS CLASS SHOULD NEVER BE MODIFIED

    properties (GetAccess = private, SetAccess = private)
        previous_timestep
        checked_poses_already = false
        called_step_already = true
        x_lin_vel_coef = 0.86;
        y_lin_vel_coef = 0.81;
        ang_vel_coef = 0.46;
    end

    methods

        function this = Robotarium(number_of_agents, save_data,
initial_poses)

```

```

        this = this@ARobotarium(number_of_agents, save_data,
initial_poses);
        this.previous_timestep = tic;

        this.initialize_visualization()
    end

    function poses = get_poses(this)

        assert(~this.checked_poses_already, 'Can only call
get_poses() once per call of step()!');

        poses = this.poses;

        %Include delay to mimic behavior of real system
        this.previous_timestep = tic;

        %Make sure it's only called once per iteration
        this.checked_poses_already = true;
        this.called_step_already = false;
    end

    function step(this)

        assert(~this.called_step_already, 'Make sure you call
get_poses before calling step!');

        %Vectorize update to states
        i = 1:this.number_of_agents;

        total_time = this.time_step + max(0,
toc(this.previous_timestep) - this.time_step);

        %Update velocities using unicycle dynamics
        this.poses(1, i) = this.poses(1, i) +
this.x_lin_vel_coef*total_time.*this.velocities(1,
i).*cos(this.poses(3, i));
        this.poses(2, i) = this.poses(2, i) +
this.y_lin_vel_coef*total_time.*this.velocities(1,
i).*sin(this.poses(3, i));
        this.poses(3, i) = this.poses(3, i) +
this.ang_vel_coef*total_time.*this.velocities(2, i);

        %Ensure that we're in the right range
        this.poses(3, i) = atan2(sin(this.poses(3, i)),
cos(this.poses(3, i)));

        %Allow getting of poses again
        this.checked_poses_already = false;
        this.called_step_already = true;

        if(this.save_data)
            this.save();
        end
    end

```

```

        this.draw_robots();
    end

    function call_at_scripts_end(this)
        if(this.save_data)
            this.mat_file_path.robotarium_data =
this.mat_file_path.robotarium_data(:, 1:(this.current_saved_ite-
1));
        end
    end
end
end
end

```

B.1.3 Microscopic

This microscopic modeling section is differently organized since the microscopic simulation does not happen in Matlab itself but in the ROS/Gazebo framework. Hence, this section mostly contains scripts used to generate proper simulation files for ROS and Gazebo, as well as the scripts necessary to parse the generated results files. The ROS/Gazebo files are later given in APPENDIX C page 622.

generateLaunchFile.m

Generates a launch file to be used directly by the command *roslaunch* to run the consensus mission with the microscopic model.

```

function generateLaunchFile(v,N,initialPositions)
% Create launch file
if isunix
    filename = sprintf('launch/robotarium_%d_%3.2f.launch',N,v);
else
    filename = sprintf('launch/robotarium %d %3.2f.launch',N,v);
end
fileID = fopen(filename, 'w');

% Add headers
fprintf(fileID, '<?xml version="1.0"?>');fprintf(fileID, '\n');
fprintf(fileID, '<launch>');fprintf(fileID, '\n');
fprintf(fileID, '    <!-- Environment -->');fprintf(fileID, '\n');

```

```

fprintf(fileID, '    <include file="$(find
gazebo_ros)/launch/empty_world.launch">'); fprintf(fileID, '\n');
fprintf(fileID, '        <arg name="world_name" value="$(find
gritbot_gazebo)/worlds/robotarium.world"/>'); fprintf(fileID, '\n');
fprintf(fileID, '        <arg name="paused"
value="false"/>'); fprintf(fileID, '\n');
fprintf(fileID, '        <arg name="use_sim_time"
value="true"/>'); fprintf(fileID, '\n');
fprintf(fileID, '        <arg name="gui"
value="true"/>'); fprintf(fileID, '\n');
fprintf(fileID, '        <arg name="headless"
value="false"/>'); fprintf(fileID, '\n');
fprintf(fileID, '        <arg name="debug"
value="false"/>'); fprintf(fileID, '\n');
fprintf(fileID, '    </include>'); fprintf(fileID, '\n');
fprintf(fileID, ''); fprintf(fileID, '\n');

% Add robots
fprintf(fileID, '    <!-- Robots -->'); fprintf(fileID, '\n');
for i = 1:N
    fprintf(fileID, '        <node name="gritbot%d" ns="gritbot%d"
pkg="gazebo_ros" type="spawn_model" args="-file $(find
gritbot_description)/urdf/gritbot_%d.urdf -urdf -x %3.2f -y %3.2f -z 0
-Y %3.2f -model gritbot%d" />', ...

i, i, i, initialPositions(1, i), initialPositions(2, i), initialPositions(3, i)
, i); fprintf(fileID, '\n');
end
fprintf(fileID, '    '); fprintf(fileID, '\n');

% Add footers
fprintf(fileID, '    <!-- Robotarium tracker -->'); fprintf(fileID, '\n');
fprintf(fileID, '        <node name="tracker" ns="robotarium"
pkg="gritbot_navigation" type="tracker" />'); fprintf(fileID, '\n');
fprintf(fileID, ''); fprintf(fileID, '\n');
fprintf(fileID, '    <!-- Data logger -->'); fprintf(fileID, '\n');
fprintf(fileID, '        <node name="logger" ns="robotarium"
pkg="gritbot_navigation" type="logger"
args="/home/jdurand7/Dropbox/phd/code/modeling/microscopic/logs/log_%d
%3.2f.csv" />', N, v); fprintf(fileID, '\n');
fprintf(fileID, '    '); fprintf(fileID, '\n');
fprintf(fileID, '    <!-- Static consensus algorithm --
>'); fprintf(fileID, '\n');
fprintf(fileID, '    <!-- NOTE: this node is required, if it stops, the
whole simulation stops -->'); fprintf(fileID, '\n');
fprintf(fileID, '        <node name="consensus" ns="robotarium"
pkg="gritbot_navigation" type="consensus" args="%3.2f" output="screen"
required="true"/>', v); fprintf(fileID, '\n');
fprintf(fileID, '</launch>');

% Close file
fclose(fileID);
end

```

Example of generated file: robotarium_3_0.02.launch

```
<?xml version="1.0"?>
<launch>
  <!-- Environment -->
  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="world_name" value="$(find
gritbot_gazebo)/worlds/robotarium.world"/>
    <arg name="paused" value="false"/>
    <arg name="use_sim_time" value="true"/>
    <arg name="gui" value="true"/>
    <arg name="headless" value="false"/>
    <arg name="debug" value="false"/>
  </include>

  <!-- Robots -->
  <node name="gritbot1" ns="gritbot1" pkg="gazebo_ros"
type="spawn_model" args="-file $(find
gritbot_description)/urdf/gritbot_1.urdf -urdf -x 0.10 -y 0.05 -z 0 -Y
1.53 -model gritbot1" />
  <node name="gritbot2" ns="gritbot2" pkg="gazebo_ros"
type="spawn_model" args="-file $(find
gritbot_description)/urdf/gritbot_2.urdf -urdf -x -0.20 -y 0.05 -z 0 -Y
2.08 -model gritbot2" />
  <node name="gritbot3" ns="gritbot3" pkg="gazebo_ros"
type="spawn_model" args="-file $(find
gritbot_description)/urdf/gritbot_3.urdf -urdf -x 0.00 -y -0.15 -z 0 -Y
5.55 -model gritbot3" />

  <!-- Robotarium tracker -->
  <node name="tracker" ns="robotarium" pkg="gritbot_navigation"
type="tracker" />

  <!-- Data logger -->
  <node name="logger" ns="robotarium" pkg="gritbot_navigation"
type="logger"
args="/home/jdurand7/Dropbox/phd/code/modeling/microscopic/logs/log_3_0
.02.csv" />

  <!-- Static consensus algorithm -->
  <!-- NOTE: this node is required, if it stops, the whole simulation
stops -->
  <node name="consensus" ns="robotarium" pkg="gritbot_navigation"
type="consensus" args="0.02" output="screen" required="true"/>
</launch>
```

doe_microscopic.m

```
% Prepare workspace
clc
close all
clear
```



```

%% Design space
Nvec = 2:5;
Vvec = .01:0.01:0.1;

[N,V] = meshgrid(Nvec,Vvec);

% Analysis
T_micro = zeros(size(N));
X_micro = cell(size(N));
t_micro = zeros(size(N));
n1 = size(N,1);
n2 = size(N,2);

%% 1) Generate launch files
for i = 1:n1
    for j = 1:n2
        fprintf('Generating launch file %d/%d (%.2f%%)\n', (i-1)*n2 +
j,n1*n2,100*((i-1)*n2 + j)/(n1*n2))

        % Load initial positions
load(sprintf('../initialPoses/init_poses_%d_%.2f.mat',N(i,j),V(i,j)))

        % Create corresponding ROS launch file
generateLaunchFile(V(i,j),N(i,j),x0);
    end
end

% 2) Run batch ROS manually here
% roscore
% sh
/home/jdurand7/Dropbox/phd/code/plots/modeling/launch/process_batch.sh

% 3) Parse results

```

process_batch.sh

A small shell script which runs all the required launch files contained in the experiments folder, one by one.

```

FILES=/home/jdurand7/Dropbox/phd/code/modeling/microscopic/launch/*.lau
nch

# Initialize log file
> /home/jdurand7/Dropbox/phd/code/modeling/microscopic/logs/log.txt

for f in $FILES
do
    # Iterate
    i=$((i+1))

```

```

# Display info
echo ""
echo "#####"
echo "Running Case $i"
echo "File $f"
echo ""

# Run ROS case
echo $f >>
/home/jdurand7/Dropbox/phd/code/modeling/microscopic/logs/log.txt
date +%s.%N >>
/home/jdurand7/Dropbox/phd/code/modeling/microscopic/logs/log.txt
roslaunch $f
date +%s.%N >>
/home/jdurand7/Dropbox/phd/code/modeling/microscopic/logs/log.txt
done

```

parseLogs.m

A high-level script which parses the logs from all experiments and saves the results in a MAT-file just like the macroscopic and mesoscopic model experiments.

```

% Prepare workspace
clc
close all
clear

%% Design space
Nvec = 2:5;
Vvec = .01:0.01:0.1;

[N,V] = meshgrid(Nvec,Vvec);

%% Analysis
T_micro = zeros(size(N));
X_micro = cell(size(N));
n1 = size(N,1);
n2 = size(N,2);

% Load data about runtimes
data = importlogfile('logs/log.txt');
% Compute total run time for each experiment
time = zeros(n1,n2);%data(3:3:end) - data(2:3:end);
% Reformat data
t_micro = reshape(time,n1,n2);

% Parse results files based on the DOE
for i = 1:n1
    for j = 1:n2
        fprintf('Parsing output file %d/%d (%.2f%%)\n', (i-1)*n2 +
j,n1*n2,100*((i-1)*n2 + j)/(n1*n2))
    end
end

```

```

        % Perform model analysis
        [t,x] =
parseOutputFile(sprintf('logs/log_%d_%3.2f.csv',N(i,j),V(i,j)));
        T_micro(i,j) = t;
        X_micro{i,j} = x;
    end
end

save('microscopic.mat')

```

importlogfile.m

A parser which reads the *log.txt* file containing the starting and ending times of each experiment.

```

function log1 = importlogfile(filename, startRow, endRow)
%IMPORTFILE Import numeric data from a text file as column vectors.
% LOG1 = IMPORTFILE(FILENAME) Reads data from text file FILENAME for
the
% default selection.
%
% LOG1 = IMPORTFILE(FILENAME, STARTROW, ENDROW) Reads data from rows
% STARTROW through ENDROW of text file FILENAME.
%
% Example:
% log1 = importfile('log.txt',1, 120);
%
% See also TEXTSCAN.

% Auto-generated by MATLAB on 2016/11/14 18:40:35

%% Initialize variables.
delimiter = ';';
if nargin<=2
    startRow = 1;
    endRow = inf;
end

%% Read columns of data as strings:
% For more information, see the TEXTSCAN documentation.
formatSpec = '%s%[\n\r]';

%% Open the text file.
fileID = fopen(filename,'r');

%% Read columns of data according to format string.
% This call is based on the structure of the file used to generate this
% code. If an error occurs for a different file, try regenerating the
code
% from the Import Tool.

```

```

dataArray = textscan(fileID, formatSpec, endRow(1)-startRow(1)+1,
'Delimiter', delimiter, 'HeaderLines', startRow(1)-1, 'ReturnOnError',
false);
for block=2:length(startRow)
    frewind(fileID);
    dataArrayBlock = textscan(fileID, formatSpec, endRow(block)-
startRow(block)+1, 'Delimiter', delimiter, 'HeaderLines',
startRow(block)-1, 'ReturnOnError', false);
    dataArray{1} = [dataArray{1};dataArrayBlock{1}];
end

%% Close the text file.
fclose(fileID);

%% Convert the contents of columns containing numeric strings to
numbers.
% Replace non-numeric strings with NaN.
row = repmat({''},length(dataArray{1}),length(dataArray)-1);
for col=1:length(dataArray)-1
    row(1:length(dataArray{col}),col) = dataArray{col};
end
numericData = NaN(size(dataArray{1},1),size(dataArray,2));

% Converts strings in the input cell array to numbers. Replaced non-
numeric
% strings with NaN.
rowData = dataArray{1};
for row=1:size(rowData, 1);
    % Create a regular expression to detect and remove non-numeric
prefixes and
    % suffixes.
    regexstr = '(?<prefix>.*?)(?<numbers>([-
]*(\d+(\[,]*)+(\.[\,]*\d*[eEdD]{0,1}[-+]*\d*[i]{0,1})|([-
]*(\d+(\[,]*)+(\.[\,]*\d+[eEdD]{0,1}[-+]*\d*[i]{0,1}))(?<suffix>.*?));
    try
        result = regexp(rowData{row}, regexstr, 'names');
        numbers = result.numbers;

        % Detected commas in non-thousand locations.
        invalidThousandsSeparator = false;
        if any(numbers==',' );
            thousandsRegExp = '^(\d+(\, \d{3})*\.[\,]*\d*$)';
            if isempty(regexp(numbers, thousandsRegExp, 'once'));
                numbers = NaN;
                invalidThousandsSeparator = true;
            end
        end
        % Convert numeric strings to numbers.
        if ~invalidThousandsSeparator;
            numbers = textscan(strrep(numbers, ',', ''), '%f');
            numericData(row, 1) = numbers{1};
            row{row, 1} = numbers{1};
        end
    catch me
    end
end
end

```

```

%% Replace non-numeric cells with NaN
R = cellfun(@(x) ~isnumeric(x) && ~islogical(x),raw); % Find non-
numeric cells
raw(R) = {NaN}; % Replace non-numeric cells

%% Allocate imported array to column variable names
log1 = cell2mat(raw(:, 1));

```

parseOutputFile.m

A function parsing log files from a given experiment, it directly outputs the consensus time as well as the consensus location after analyzing the log.

```

function [t,x] = parseOutputFile(filename)
% Read file data
M = csvread(filename);

% Final position
x = [mean(M(end,2:2:end-1)); mean(M(end,3:2:end))];

% Time to reach consensus
i = 100;
count = 0;
d = 0;
d_prev = 1;
while i < size(M,1) && count < 25
    if abs(d-d_prev) < 1e-5
        count = count + 1;
    else
        count = 0;
    end

    d_prev = d;
    dx = M(i,2:2:end-1) - M(i-1,2:2:end-1);
    dy = M(i,3:2:end) - M(i-1,3:2:end);
    d = mean(sqrt(dx.^2 + dy.^2));

    % Increment
    i = i + 1;
end
t = M(i-25,1) - M(1,1);

% Last N iterations are stopping the algorithm
% i = size(M,1);
% t = M(i-25,1) - M(1,1);
end

```

Example of log file: log_3_0.02.csv

A log file generated after a ROS experiment. The first column is the simulation time and the following columns are the x, y coordinate pairs for each robot (3 in this case).

```
5.128000000,-0.2,0.05,0.1,0.05,8.64825e-16,-0.15
5.160000000,-0.2,0.05,0.1,0.05,8.64825e-16,-0.15
5.193000000,-0.2,0.05,0.1,0.05,8.64825e-16,-0.15
5.227000000,-0.2,0.05,0.1,0.05,8.64825e-16,-0.15
5.260000000,-0.2,0.05,0.1,0.05,8.64825e-16,-0.15
5.293000000,-0.2,0.05,0.1,0.05,8.64825e-16,-0.15
5.327000000,-0.2,0.05,0.1,0.05,8.64825e-16,-0.15
5.360000000,-0.199992,0.0499854,0.0999993,0.0499833,8.64825e-16,-0.15
5.393000000,-0.199695,0.0493689,0.100013,0.0492991,8.64825e-16,-0.15
5.427000000,-0.199695,0.0493689,0.100013,0.0492991,8.64825e-16,-0.15
5.460000000,-0.199487,0.0487697,0.100098,0.0486705,8.64825e-16,-0.15
5.493000000,-0.19934,0.0481185,0.100261,0.0480231,-0.000266759,-0.149774
5.527000000,-0.199269,0.0474546,0.100497,0.0473988,-0.000810799,-0.149391
5.560000000,-0.199275,0.046787,0.100803,0.0468056,-0.00139606,-0.149072
5.593000000,-0.199358,0.0461246,0.101175,0.0462511,-0.00201478,-0.148821
5.627000000,-0.199358,0.0461246,0.101175,0.0462511,-0.00201478,-0.148821
5.660000000,-0.199517,0.0454761,0.101314,0.0460622,-0.00265829,-0.148643
5.693000000,-0.19975,0.0448503,0.100824,0.0465147,-0.003318,-0.14854
5.727000000,-0.200053,0.0442552,0.100286,0.0469065,-0.00398513,-0.148515
5.761000000,-0.200385,0.0437513,0.099706,0.047234,-0.00465079,-0.148566
5.793000000,-0.199993,0.0442115,0.0990911,0.047494,-0.0053061,-0.148694
5.827000000,-0.199993,0.0442115,0.0990911,0.047494,-0.0053061,-0.148694
5.860000000,-0.199506,0.044667,0.0984503,0.0476814,-0.00594233,-0.148896
5.893000000,-0.19897,0.0450623,0.0977922,0.0477938,-0.006551,-0.14917
5.927000000,-0.198392,0.0453934,0.0971255,0.0478295,-0.00675009,-0.149279
5.960000000,-0.197779,0.0456574,0.0964592,0.0477881,-0.0063142,-0.148939
5.993000000,-0.197123,0.0458527,0.095785,0.0476711,-0.00582346,-0.148462
6.027000000,-0.197123,0.0458527,0.095785,0.0476711,-0.00582346,-0.148462
6.060000000,-0.196498,0.0459634,0.095171,0.0475105,-0.0054241,-0.147972
6.093000000,-0.195832,0.0460053,0.0945358,0.047304,-0.00506465,-0.147411
6.127000000,-0.195165,0.0459701,0.0939126,0.0470658,-0.00477038,-0.146812
6.160000000,-0.194507,0.0458594,0.0933003,0.0468096,-0.00454675,-0.146183
6.193000000,-0.193862,0.0456861,0.092691,0.046549,-0.00439711,-0.145533
6.227000000,-0.193862,0.0456861,0.092691,0.046549,-0.00439711,-0.145533
6.260000000,-0.193228,0.0454763,0.0920801,0.0462921,-0.00432332,-0.144869
6.293000000,-0.192604,0.0452529,0.0914667,0.0460415,-0.00431969,-0.144201
6.327000000,-0.191984,0.0450187,0.0908506,0.0457972,-0.00438072,-0.143536
6.360000000,-0.191366,0.0447796,0.0902326,0.0455578,-0.0044933,-0.142878
6.393000000,-0.190749,0.0445395,0.0896135,0.0453212,-0.00464209,-0.142227
6.428000000,-0.190131,0.0442998,0.088994,0.0450856,-0.00481243,-0.141583
6.460000000,-0.190131,0.0442998,0.088994,0.0450856,-0.00481243,-0.141583
6.493000000,-0.189513,0.0440608,0.0883744,0.0448502,-0.00499547,-0.140946
6.527000000,-0.188894,0.0438221,0.0877546,0.0446156,-0.00518965,-0.140312
6.561000000,-0.188276,0.0435834,0.0871343,0.044382,-0.00539392,-0.139681
6.593000000,-0.187658,0.0433446,0.0865133,0.0441503,-0.00560674,-0.139054
6.627000000,-0.187024,0.0430999,0.0858759,0.0439157,-0.00583261,-0.138413
6.660000000,-0.187024,0.0430999,0.0858759,0.0439157,-0.00583261,-0.138413
6.693000000,-0.186437,0.0428727,0.0852839,0.0437012,-0.00604794,-0.137821
6.727000000,-0.185819,0.0426334,0.0846595,0.0434788,-0.00628066,-0.137201
6.760000000,-0.185201,0.042394,0.0840338,0.0432599,-0.00651937,-0.136582
6.793000000,-0.184583,0.0421544,0.0834068,0.0430448,-0.00676381,-0.135966
6.827000000,-0.183965,0.0419147,0.0827785,0.0428336,-0.00701369,-0.135353
6.860000000,-0.183965,0.0419147,0.0827785,0.0428336,-0.00701369,-0.135353
6.893000000,-0.183347,0.0416751,0.0821488,0.0426263,-0.00726878,-0.134741
```

6.927000000,-0.182729,0.0414354,0.0815179,0.042423,-0.00752894,-0.134131
6.960000000,-0.182111,0.0411957,0.0808856,0.0422237,-0.00779411,-0.133524
6.993000000,-0.181493,0.040956,0.0802521,0.0420283,-0.00806415,-0.132919
7.027000000,-0.180876,0.0407163,0.0796173,0.041837,-0.00833895,-0.132316
7.060000000,-0.180876,0.0407163,0.0796173,0.041837,-0.00833895,-0.132316
7.093000000,-0.180258,0.0404767,0.0789814,0.0416495,-0.00861838,-0.131715
7.127000000,-0.17964,0.0402371,0.0783443,0.041466,-0.00890236,-0.131117
7.160000000,-0.179022,0.0399975,0.077706,0.0412864,-0.00919084,-0.13052
7.194000000,-0.178388,0.039752,0.0770506,0.0411063,-0.00949109,-0.129911
7.227000000,-0.177801,0.0395244,0.0764422,0.0409431,-0.00977345,-0.129349
7.260000000,-0.177801,0.0395244,0.0764422,0.0409431,-0.00977345,-0.129349
7.293000000,-0.177183,0.0392849,0.0758008,0.0407749,-0.0100748,-0.128759
7.327000000,-0.176565,0.0390455,0.0751584,0.0406104,-0.0103803,-0.128171
7.360000000,-0.175947,0.0388061,0.0745151,0.0404497,-0.01069,-0.127585
7.393000000,-0.175329,0.0385668,0.0738708,0.0402927,-0.0110037,-0.127002
7.427000000,-0.174711,0.0383275,0.0732256,0.0401393,-0.0113213,-0.12642
7.460000000,-0.174711,0.0383275,0.0732256,0.0401393,-0.0113213,-0.12642
7.493000000,-0.174093,0.0380882,0.0725795,0.0399896,-0.0116429,-0.125841
7.527000000,-0.173475,0.037849,0.0719327,0.0398433,-0.0119683,-0.125264
7.560000000,-0.172857,0.0376098,0.071285,0.0397006,-0.0122975,-0.124689
7.593000000,-0.172239,0.0373706,0.0706365,0.0395614,-0.0126304,-0.124117
7.627000000,-0.171621,0.0371315,0.0699873,0.0394256,-0.0129669,-0.123546
7.693000000,-0.171621,0.0371315,0.0699873,0.0394256,-0.0129669,-0.123546
7.727000000,-0.170384,0.0366535,0.0686868,0.039164,-0.0136505,-0.122411
7.760000000,-0.169751,0.0364093,0.0680191,0.0390348,-0.0140059,-0.121833
7.793000000,-0.169162,0.0361846,0.0674,0.0389177,-0.0143373,-0.121298
7.827000000,-0.168542,0.0359511,0.0667477,0.0387966,-0.0146872,-0.120736
7.893000000,-0.168542,0.0359511,0.0667477,0.0387966,-0.0146872,-0.120736
7.927000000,-0.16792,0.035722,0.0660952,0.0386777,-0.0150374,-0.120173
7.960000000,-0.167297,0.0354976,0.0654422,0.0385605,-0.0153878,-0.119611
7.993000000,-0.166671,0.0352781,0.0647889,0.0384451,-0.015738,-0.119049
8.027000000,-0.166044,0.0350638,0.0641353,0.0383319,-0.016088,-0.118487
8.060000000,-0.165415,0.0348546,0.0634813,0.0382207,-0.0164376,-0.117924
8.093000000,-0.165415,0.0348546,0.0634813,0.0382207,-0.0164376,-0.117924
8.127000000,-0.164784,0.0346506,0.0628268,0.0381117,-0.0167868,-0.117361
8.160000000,-0.164152,0.0344521,0.0621721,0.0380051,-0.0171356,-0.116798
8.193000000,-0.163517,0.0342589,0.0615169,0.0379007,-0.0174839,-0.116235
8.227000000,-0.162881,0.0340711,0.0608613,0.0377986,-0.0178316,-0.115671
8.260000000,-0.162244,0.0338882,0.0602054,0.037699,-0.0181789,-0.115107
8.293000000,-0.162244,0.0338882,0.0602054,0.037699,-0.0181789,-0.115107
8.327000000,-0.161605,0.033709,0.0595491,0.0376018,-0.0185256,-0.114543
8.360000000,-0.160964,0.0335327,0.0588923,0.0375071,-0.0188719,-0.113978
8.393000000,-0.160322,0.0333627,0.0582353,0.037415,-0.0192176,-0.113413
8.427000000,-0.159679,0.0332011,0.0575779,0.037325,-0.0195627,-0.112848
8.460000000,-0.159034,0.0330444,0.0569202,0.0372365,-0.019907,-0.112282
8.493000000,-0.159034,0.0330444,0.0569202,0.0372365,-0.019907,-0.112282
8.527000000,-0.158389,0.0328903,0.0562626,0.037149,-0.0202501,-0.111715
8.560000000,-0.157744,0.0327378,0.0556047,0.0370613,-0.0205917,-0.111148
8.593000000,-0.157098,0.0325864,0.054947,0.0369735,-0.0209317,-0.110579
8.627000000,-0.156452,0.0324358,0.0542894,0.0368853,-0.0212701,-0.11001
8.660000000,-0.155806,0.0322858,0.0536319,0.0367965,-0.0216067,-0.109439
8.693000000,-0.155806,0.0322858,0.0536319,0.0367965,-0.0216067,-0.109439
8.727000000,-0.155144,0.0321323,0.052958,0.0367045,-0.0219499,-0.108854
8.760000000,-0.15453,0.0319902,0.0523336,0.0366184,-0.0222663,-0.10831
8.793000000,-0.153884,0.0318407,0.0516764,0.0365268,-0.0225975,-0.107736
8.827000000,-0.153238,0.0316913,0.0510194,0.0364341,-0.0229268,-0.107161
8.861000000,-0.152592,0.0315419,0.0503625,0.0363402,-0.0232543,-0.106585
8.893000000,-0.152592,0.0315419,0.0503625,0.0363402,-0.0232543,-0.106585
8.927000000,-0.151946,0.0313924,0.0497058,0.0362452,-0.02358,-0.106008
8.960000000,-0.1513,0.0312429,0.0490494,0.0361491,-0.0239039,-0.105431
8.993000000,-0.150653,0.0310934,0.0483931,0.0360519,-0.0242261,-0.104852
9.027000000,-0.150007,0.0309437,0.047737,0.0359534,-0.0245464,-0.104272
9.060000000,-0.149361,0.030794,0.0470811,0.0358537,-0.024865,-0.103691

9.093000000,-0.149361,0.030794,0.0470811,0.0358537,-0.024865,-0.103691
9.127000000,-0.148715,0.0306442,0.0464253,0.0357528,-0.0251818,-0.103109
9.160000000,-0.148069,0.0304943,0.0457696,0.0356507,-0.025497,-0.102527
9.193000000,-0.147423,0.0303443,0.0451143,0.0355475,-0.0258105,-0.101943
9.227000000,-0.146777,0.0301941,0.0444592,0.0354431,-0.0261223,-0.101359
9.260000000,-0.146131,0.0300439,0.0438042,0.0353376,-0.0264326,-0.100773
9.294000000,-0.146131,0.0300439,0.0438042,0.0353376,-0.0264326,-0.100773
9.327000000,-0.145485,0.0298935,0.0431495,0.0352309,-0.0267412,-0.100187
9.360000000,-0.144839,0.029743,0.0424949,0.035123,-0.0270483,-0.0996
9.394000000,-0.144177,0.0295887,0.0418242,0.0350112,-0.0273614,-0.0989976
9.427000000,-0.143564,0.0294455,0.0412027,0.0349066,-0.0276503,-0.0984385
9.460000000,-0.142918,0.0292947,0.0405487,0.0347953,-0.027953,-0.0978491
9.493000000,-0.142918,0.0292947,0.0405487,0.0347953,-0.027953,-0.0978491
9.527000000,-0.142272,0.0291438,0.0398949,0.034683,-0.0282431,-0.0972806
9.560000000,-0.141627,0.0289927,0.0392413,0.0345696,-0.0285121,-0.0967502
9.593000000,-0.140981,0.0288416,0.0385879,0.0344551,-0.028775,-0.0962288
9.627000000,-0.140335,0.0286903,0.0379347,0.0343395,-0.0290186,-0.0957426
9.660000000,-0.139689,0.0285389,0.0372817,0.0342229,-0.0292457,-0.0952866
9.693000000,-0.139689,0.0285389,0.0372817,0.0342229,-0.0292457,-0.0952866
9.727000000,-0.139044,0.0283873,0.0366289,0.0341052,-0.0294576,-0.0948586
9.760000000,-0.138398,0.0282355,0.0359763,0.0339865,-0.0296561,-0.0944552
9.793000000,-0.137753,0.0280835,0.0353238,0.0338667,-0.0298505,-0.0940579
9.827000000,-0.137107,0.0279312,0.0346716,0.0337458,-0.0300333,-0.0936822
9.860000000,-0.136462,0.0277786,0.0340409,0.033628,-0.0302056,-0.0933261
9.893000000,-0.136462,0.0277786,0.0340409,0.033628,-0.0302056,-0.0933261
9.927000000,-0.13583,0.027629,0.0334653,0.0335198,-0.0303677,-0.0929893
9.960000000,-0.135247,0.0274906,0.0329414,0.0334206,-0.0305205,-0.0926699
9.993000000,-0.13468,0.0273556,0.0324316,0.0333234,-0.0306702,-0.0923553
10.027000000,-0.134161,0.0272323,0.0319646,0.0332338,-0.0308117,-0.0920566
10.060000000,-0.133684,0.0271186,0.0315333,0.0331505,-0.0309456,-0.0917722
10.094000000,-0.133684,0.0271186,0.0315333,0.0331505,-0.0309456,-0.0917722
10.127000000,-0.133246,0.0270139,0.0311349,0.0330732,-0.0310722,-0.0915017
10.160000000,-0.13284,0.0269171,0.0307653,0.033001,-0.0311922,-0.0912443
10.193000000,-0.132443,0.026822,0.0304027,0.0329297,-0.03131,-0.0909901
10.227000000,-0.132072,0.0267332,0.0300632,0.0328625,-0.0314221,-0.0907471
10.260000000,-0.131726,0.02665,0.0297444,0.0327991,-0.0315288,-0.0905147
10.293000000,-0.131726,0.02665,0.0297444,0.0327991,-0.0315288,-0.0905147
10.327000000,-0.1314,0.0265719,0.0294441,0.0327391,-0.0316305,-0.0902919
10.360000000,-0.131086,0.0264963,0.0291527,0.0326806,-0.0317303,-0.0900725
10.393000000,-0.130799,0.0264273,0.0288868,0.0326268,-0.0318212,-0.0898715
10.427000000,-0.13051,0.0263575,0.0286176,0.0325721,-0.0319139,-0.0896658
10.460000000,-0.130237,0.0262917,0.028363,0.0325202,-0.0320022,-0.0894686
10.493000000,-0.130237,0.0262917,0.028363,0.0325202,-0.0320022,-0.0894686
10.527000000,-0.129979,0.0262295,0.0281213,0.0324707,-0.0320868,-0.0892791
10.560000000,-0.129735,0.0261706,0.0278918,0.0324235,-0.0321676,-0.089097
10.593000000,-0.129494,0.0261124,0.0276652,0.0323767,-0.0322474,-0.0889167
10.627000000,-0.129265,0.026057,0.0274489,0.0323318,-0.032324,-0.0887427
10.660000000,-0.129047,0.0260042,0.0272417,0.0322887,-0.0323977,-0.0885746
10.693000000,-0.128836,0.0259533,0.0270419,0.0322469,-0.032469,-0.0884112
10.727000000,-0.128836,0.0259533,0.0270419,0.0322469,-0.032469,-0.0884112
10.760000000,-0.128636,0.0259048,0.0268507,0.0322069,-0.0325376,-0.0882535
10.793000000,-0.128443,0.0258582,0.0266663,0.0321681,-0.0326039,-0.0881004
10.827000000,-0.128253,0.0258123,0.0264849,0.0321298,-0.0326691,-0.0879492
10.860000000,-0.128071,0.0257682,0.0263101,0.0320929,-0.032732,-0.0878026
10.893000000,-0.127896,0.0257259,0.0261418,0.0320572,-0.0327928,-0.0876605
10.927000000,-0.127896,0.0257259,0.0261418,0.0320572,-0.0327928,-0.0876605
10.960000000,-0.127728,0.0256851,0.0259792,0.0320226,-0.0328516,-0.0875223
10.993000000,-0.127565,0.0256459,0.0258222,0.0319891,-0.0329086,-0.087388
11.027000000,-0.127405,0.025607,0.0256667,0.0319558,-0.0329649,-0.0872548
11.060000000,-0.12725,0.0255695,0.0255161,0.0319235,-0.0330195,-0.0871251
11.093000000,-0.1271,0.0255333,0.0253705,0.0318922,-0.0330724,-0.086999
11.128000000,-0.1271,0.0255333,0.0253705,0.0318922,-0.0330724,-0.086999
11.160000000,-0.126952,0.0254975,0.0252259,0.0318611,-0.033125,-0.0868732

11.193000000,-0.126819,0.0254654,0.025096,0.0318331,-0.0331723,-0.0867595
11.227000000,-0.12668,0.0254319,0.0249603,0.0318037,-0.0332216,-0.0866407
11.260000000,-0.126545,0.0253994,0.0248287,0.0317752,-0.0332695,-0.0865248
11.293000000,-0.126415,0.025368,0.0247007,0.0317474,-0.033316,-0.0864118
11.327000000,-0.126415,0.025368,0.0247007,0.0317474,-0.033316,-0.0864118
11.360000000,-0.126289,0.0253375,0.0245763,0.0317204,-0.0333614,-0.0863014
11.393000000,-0.126166,0.025308,0.0244554,0.0316941,-0.0334054,-0.0861936
11.427000000,-0.126044,0.0252787,0.0243354,0.0316679,-0.033449,-0.0860866
11.460000000,-0.125926,0.0252503,0.0242187,0.0316424,-0.0334915,-0.085982
11.493000000,-0.125811,0.0252227,0.024105,0.0316175,-0.0335329,-0.0858798
11.527000000,-0.125811,0.0252227,0.024105,0.0316175,-0.0335329,-0.0858798
11.560000000,-0.125699,0.0251958,0.0239938,0.0315932,-0.0335733,-0.0857795
11.593000000,-0.12559,0.0251697,0.0238857,0.0315695,-0.0336126,-0.0856816
11.627000000,-0.125483,0.0251439,0.0237787,0.031546,-0.0336515,-0.0855846
11.660000000,-0.125378,0.0251188,0.0236742,0.031523,-0.0336895,-0.0854896
11.694000000,-0.125275,0.0250943,0.0235722,0.0315006,-0.0337265,-0.0853965
11.727000000,-0.125275,0.0250943,0.0235722,0.0315006,-0.0337265,-0.0853965
11.760000000,-0.125176,0.0250706,0.0234728,0.0314787,-0.0337625,-0.0853055
11.793000000,-0.125079,0.0250475,0.0233758,0.0314574,-0.0337977,-0.0852164
11.827000000,-0.124982,0.0250245,0.0232793,0.0314361,-0.0338327,-0.0851277
11.860000000,-0.124888,0.0250021,0.0231851,0.0314153,-0.0338668,-0.0850409
11.893000000,-0.124794,0.0249798,0.0230908,0.0313944,-0.0339009,-0.0849537
11.927000000,-0.124794,0.0249798,0.0230908,0.0313944,-0.0339009,-0.0849537
11.961000000,-0.124709,0.0249596,0.0230053,0.0313756,-0.0339319,-0.0848744
11.993000000,-0.124622,0.0249389,0.0229173,0.0313562,-0.0339637,-0.0847926
12.027000000,-0.124535,0.0249183,0.0228298,0.0313368,-0.0339952,-0.0847112
12.060000000,-0.12445,0.0248982,0.0227441,0.0313178,-0.0340261,-0.0846312
12.093000000,-0.124367,0.0248786,0.0226603,0.0312993,-0.0340563,-0.0845528
12.127000000,-0.124367,0.0248786,0.0226603,0.0312993,-0.0340563,-0.0845528
12.160000000,-0.124286,0.0248595,0.0225782,0.0312811,-0.0340858,-0.0844759
12.193000000,-0.124206,0.0248408,0.0224978,0.0312633,-0.0341147,-0.0844003
12.227000000,-0.124127,0.0248223,0.0224179,0.0312456,-0.0341434,-0.0843251
12.260000000,-0.12405,0.0248041,0.0223395,0.0312283,-0.0341715,-0.0842512
12.293000000,-0.123974,0.0247864,0.0222627,0.0312113,-0.034199,-0.0841787
12.327000000,-0.123974,0.0247864,0.0222627,0.0312113,-0.034199,-0.0841787
12.360000000,-0.1239,0.0247691,0.0221873,0.0311945,-0.034226,-0.0841073
12.393000000,-0.123827,0.0247521,0.0221133,0.0311782,-0.0342524,-0.0840371
12.427000000,-0.123755,0.0247352,0.0220398,0.0311619,-0.0342786,-0.0839673
12.460000000,-0.123684,0.0247188,0.0219677,0.0311459,-0.0343043,-0.0838986
12.493000000,-0.123684,0.0247188,0.0219677,0.0311459,-0.0343043,-0.0838986
12.527000000,-0.123608,0.0247011,0.02189,0.0311287,-0.034332,-0.0838245
12.560000000,-0.123537,0.0246846,0.0218173,0.0311126,-0.0343577,-0.0837552
12.593000000,-0.123473,0.0246698,0.0217519,0.0310981,-0.034381,-0.0836925
12.628000000,-0.123407,0.0246545,0.0216845,0.0310832,-0.0344049,-0.0836278
12.660000000,-0.123342,0.0246396,0.0216182,0.0310685,-0.0344284,-0.0835642
12.693000000,-0.123342,0.0246396,0.0216182,0.0310685,-0.0344284,-0.0835642
12.727000000,-0.123277,0.0246246,0.0215515,0.0310538,-0.0344518,-0.0835001
12.760000000,-0.123217,0.024611,0.0214908,0.0310404,-0.0344732,-0.0834416
12.793000000,-0.123155,0.0245968,0.0214271,0.0310263,-0.0344957,-0.0833801
12.827000000,-0.123094,0.0245829,0.0213645,0.0310125,-0.0345177,-0.0833196
12.860000000,-0.123034,0.0245692,0.021303,0.030999,-0.0345393,-0.0832601
12.893000000,-0.123034,0.0245692,0.021303,0.030999,-0.0345393,-0.0832601
12.927000000,-0.122975,0.0245558,0.0212425,0.0309856,-0.0345605,-0.0832014
12.960000000,-0.122918,0.0245427,0.0211831,0.0309725,-0.0345814,-0.0831436
12.993000000,-0.12286,0.0245296,0.0211238,0.0309595,-0.0346021,-0.0830859
13.027000000,-0.122803,0.0245167,0.0210652,0.0309466,-0.0346226,-0.083029
13.060000000,-0.122747,0.0245041,0.0210076,0.0309339,-0.0346426,-0.0829728
13.093000000,-0.122747,0.0245041,0.0210076,0.0309339,-0.0346426,-0.0829728
13.127000000,-0.122692,0.0244918,0.0209511,0.0309215,-0.0346623,-0.0829176
13.160000000,-0.122638,0.0244797,0.0208957,0.0309093,-0.0346816,-0.0828633
13.193000000,-0.122585,0.0244677,0.0208405,0.0308972,-0.0347007,-0.0828093
13.228000000,-0.122532,0.024456,0.0207862,0.0308853,-0.0347195,-0.082756
13.260000000,-0.122481,0.0244444,0.0207327,0.0308736,-0.034738,-0.0827035

13.293000000,-0.122481,0.0244444,0.0207327,0.0308736,-0.034738,-0.0827035
13.327000000,-0.12243,0.0244331,0.0206801,0.0308621,-0.0347562,-0.0826517
13.361000000,-0.122379,0.024422,0.0206283,0.0308508,-0.034774,-0.0826006
13.393000000,-0.12233,0.024411,0.0205767,0.0308395,-0.0347918,-0.0825497
13.427000000,-0.12228,0.0244001,0.0205258,0.0308285,-0.0348093,-0.0824995
13.460000000,-0.122232,0.0243895,0.0204757,0.0308176,-0.0348264,-0.08245
13.493000000,-0.122232,0.0243895,0.0204757,0.0308176,-0.0348264,-0.08245
13.527000000,-0.122184,0.0243791,0.0204264,0.0308068,-0.0348433,-0.0824011
13.560000000,-0.122138,0.0243688,0.0203778,0.0307963,-0.0348599,-0.0823528
13.593000000,-0.122091,0.0243586,0.0203294,0.0307858,-0.0348765,-0.0823048
13.627000000,-0.122044,0.0243483,0.0202804,0.0307752,-0.0348932,-0.082256
13.660000000,-0.122001,0.024339,0.0202357,0.0307655,-0.0349083,-0.0822116
13.693000000,-0.122001,0.024339,0.0202357,0.0307655,-0.0349083,-0.0822116
13.727000000,-0.121956,0.0243293,0.0201893,0.0307555,-0.0349241,-0.0821654
13.760000000,-0.121912,0.0243198,0.0201436,0.0307457,-0.0349395,-0.0821197
13.793000000,-0.121868,0.0243104,0.020098,0.0307358,-0.034955,-0.0820742
13.827000000,-0.121825,0.0243011,0.0200531,0.0307262,-0.0349701,-0.0820293
13.860000000,-0.121782,0.0242919,0.0200087,0.0307166,-0.034985,-0.0819849
13.893000000,-0.12174,0.0242829,0.0199648,0.0307072,-0.0349998,-0.0819409
13.927000000,-0.12174,0.0242829,0.0199648,0.0307072,-0.0349998,-0.0819409
13.960000000,-0.121698,0.0242741,0.0199215,0.030698,-0.0350143,-0.0818974
13.993000000,-0.121657,0.0242654,0.0198788,0.0306888,-0.0350286,-0.0818545
14.027000000,-0.121617,0.0242568,0.0198364,0.0306798,-0.0350428,-0.0818119
14.060000000,-0.121576,0.0242483,0.0197946,0.0306709,-0.0350567,-0.0817697
14.094000000,-0.121537,0.02424,0.0197532,0.0306621,-0.0350705,-0.081728
14.127000000,-0.121537,0.02424,0.0197532,0.0306621,-0.0350705,-0.081728
14.160000000,-0.121498,0.0242317,0.0197123,0.0306534,-0.0350841,-0.0816868
14.193000000,-0.121459,0.0242237,0.019672,0.0306449,-0.0350975,-0.081646
14.227000000,-0.121421,0.0242157,0.019632,0.0306364,-0.0351107,-0.0816055
14.260000000,-0.121383,0.0242078,0.0195924,0.0306281,-0.0351238,-0.0815655
14.293000000,-0.121345,0.0242001,0.0195533,0.0306198,-0.0351367,-0.0815258
14.327000000,-0.121345,0.0242001,0.0195533,0.0306198,-0.0351367,-0.0815258
14.361000000,-0.121308,0.0241924,0.0195147,0.0306117,-0.0351493,-0.0814867
14.393000000,-0.121272,0.0241849,0.0194765,0.0306037,-0.0351619,-0.0814478
14.427000000,-0.121235,0.0241775,0.0194385,0.0305957,-0.0351743,-0.0814092
14.460000000,-0.121199,0.0241701,0.019401,0.0305879,-0.0351866,-0.081371
14.493000000,-0.121164,0.0241629,0.0193639,0.0305802,-0.0351987,-0.0813332
14.527000000,-0.121163,0.0241627,0.019363,0.03058,-0.035199,-0.0813323
14.560000000,-0.12113,0.024156,0.0193282,0.0305727,-0.0352103,-0.0812968
14.594000000,-0.121095,0.024149,0.019292,0.0305652,-0.0352221,-0.0812598
14.627000000,-0.121061,0.024142,0.0192559,0.0305577,-0.0352338,-0.0812229
14.660000000,-0.121026,0.0241352,0.0192203,0.0305504,-0.0352453,-0.0811865
14.693000000,-0.120993,0.0241285,0.0191851,0.0305431,-0.0352566,-0.0811504
14.728000000,-0.120993,0.0241285,0.0191851,0.0305431,-0.0352566,-0.0811504
14.760000000,-0.12096,0.0241218,0.0191503,0.0305359,-0.0352679,-0.0811147
14.793000000,-0.120927,0.0241153,0.0191159,0.0305289,-0.0352789,-0.0810794
14.827000000,-0.120894,0.0241088,0.0190816,0.0305219,-0.0352899,-0.0810442
14.861000000,-0.120862,0.0241025,0.0190477,0.0305149,-0.0353007,-0.0810094
14.893000000,-0.12083,0.0240962,0.0190143,0.0305081,-0.0353114,-0.0809749
14.927000000,-0.12083,0.0240962,0.0190143,0.0305081,-0.0353114,-0.0809749
14.960000000,-0.120798,0.02409,0.0189811,0.0305013,-0.035322,-0.0809408
14.993000000,-0.120767,0.0240839,0.0189484,0.0304947,-0.0353324,-0.080907
15.027000000,-0.120736,0.0240779,0.0189158,0.0304881,-0.0353427,-0.0808734
15.060000000,-0.120705,0.0240719,0.0188835,0.0304816,-0.0353529,-0.08084
15.093000000,-0.120674,0.024066,0.0188516,0.0304751,-0.035363,-0.080807
15.127000000,-0.120674,0.024066,0.0188516,0.0304751,-0.035363,-0.080807
15.160000000,-0.120644,0.0240602,0.01882,0.0304688,-0.035373,-0.0807743
15.193000000,-0.120614,0.0240544,0.018788,0.0304624,-0.035383,-0.0807411
15.227000000,-0.120586,0.024049,0.0187585,0.0304565,-0.0353924,-0.0807104
15.260000000,-0.120556,0.0240435,0.0187277,0.0304503,-0.035402,-0.0806784
15.293000000,-0.120527,0.024038,0.0186973,0.0304443,-0.0354115,-0.0806468
15.327000000,-0.120527,0.024038,0.0186973,0.0304443,-0.0354115,-0.0806468
15.360000000,-0.120498,0.0240326,0.0186671,0.0304383,-0.035421,-0.0806155

15.394000000,-0.12047,0.0240273,0.0186373,0.0304324,-0.0354303,-0.0805844
15.427000000,-0.120442,0.024022,0.0186076,0.0304265,-0.0354396,-0.0805534
15.460000000,-0.120414,0.0240167,0.0185781,0.0304207,-0.0354488,-0.0805227
15.493000000,-0.120386,0.0240116,0.018549,0.030415,-0.0354578,-0.0804923
15.527000000,-0.120386,0.0240116,0.018549,0.030415,-0.0354578,-0.0804923
15.560000000,-0.120358,0.0240065,0.0185202,0.0304094,-0.0354668,-0.0804622
15.593000000,-0.120331,0.0240016,0.0184918,0.0304038,-0.0354757,-0.0804324
15.627000000,-0.120304,0.0239966,0.0184634,0.0303983,-0.0354845,-0.0804027
15.660000000,-0.120277,0.0239917,0.0184352,0.0303928,-0.0354931,-0.0803733
15.693000000,-0.120251,0.0239869,0.0184074,0.0303874,-0.0355017,-0.0803441
15.727000000,-0.120251,0.0239869,0.0184074,0.0303874,-0.0355017,-0.0803441
15.760000000,-0.120225,0.0239822,0.0183798,0.0303821,-0.0355102,-0.0803151
15.793000000,-0.120199,0.0239775,0.0183525,0.0303769,-0.0355187,-0.0802864
15.827000000,-0.120173,0.0239729,0.0183253,0.0303717,-0.035527,-0.0802578
15.860000000,-0.120147,0.0239683,0.0182984,0.0303665,-0.0355353,-0.0802295
15.893000000,-0.120122,0.0239638,0.0182717,0.0303614,-0.0355435,-0.0802014
15.927000000,-0.120122,0.0239638,0.0182717,0.0303614,-0.0355435,-0.0802014
15.960000000,-0.120096,0.0239594,0.0182453,0.0303564,-0.0355516,-0.0801736
15.993000000,-0.120072,0.023955,0.0182192,0.0303515,-0.0355596,-0.080146
16.027000000,-0.120047,0.0239507,0.0181932,0.0303466,-0.0355676,-0.0801185
16.060000000,-0.120022,0.0239464,0.0181673,0.0303417,-0.0355755,-0.0800912
16.093000000,-0.119997,0.0239421,0.0181412,0.0303368,-0.0355835,-0.0800635
16.128000000,-0.119997,0.0239421,0.0181412,0.0303368,-0.0355835,-0.0800635
16.160000000,-0.119974,0.0239382,0.0181171,0.0303323,-0.0355908,-0.080038
16.193000000,-0.11995,0.0239341,0.0180921,0.0303277,-0.0355985,-0.0800115
16.227000000,-0.119927,0.02393,0.0180671,0.030323,-0.0356061,-0.0799849
16.260000000,-0.119903,0.023926,0.0180423,0.0303184,-0.0356136,-0.0799586
16.293000000,-0.11988,0.0239221,0.0180178,0.0303139,-0.0356211,-0.0799326
16.327000000,-0.11988,0.0239221,0.0180178,0.0303139,-0.0356211,-0.0799326
16.360000000,-0.119857,0.0239182,0.0179935,0.0303095,-0.0356284,-0.0799067
16.393000000,-0.119834,0.0239144,0.0179694,0.0303051,-0.0356357,-0.0798811
16.427000000,-0.119811,0.0239106,0.0179454,0.0303007,-0.035643,-0.0798555
16.461000000,-0.119786,0.0239065,0.0179193,0.0302959,-0.035651,-0.0798275
16.493000000,-0.119786,0.0239065,0.0179193,0.0302959,-0.035651,-0.0798275
16.527000000,-0.119762,0.0239026,0.0178945,0.0302915,-0.0356583,-0.0798013
16.560000000,-0.119741,0.0238992,0.0178724,0.0302875,-0.0356651,-0.0797775
16.593000000,-0.119719,0.0238956,0.0178491,0.0302833,-0.0356721,-0.0797526
16.627000000,-0.119697,0.023892,0.0178261,0.0302792,-0.035679,-0.079728
16.660000000,-0.119675,0.0238885,0.0178033,0.0302751,-0.0356858,-0.0797036
16.693000000,-0.119675,0.0238885,0.0178033,0.0302751,-0.0356858,-0.0797036
16.727000000,-0.119654,0.0238851,0.0177807,0.0302711,-0.0356926,-0.0796793
16.760000000,-0.119633,0.0238817,0.0177582,0.0302671,-0.0356993,-0.0796553
16.793000000,-0.119611,0.0238783,0.0177359,0.0302632,-0.035706,-0.0796312
16.827000000,-0.11959,0.023875,0.0177137,0.0302593,-0.0357127,-0.0796074
16.860000000,-0.119569,0.0238717,0.0176917,0.0302554,-0.0357192,-0.0795838
16.893000000,-0.119569,0.0238717,0.0176917,0.0302554,-0.0357192,-0.0795838
16.928000000,-0.119548,0.0238685,0.0176699,0.0302517,-0.0357257,-0.0795603
16.960000000,-0.119527,0.0238652,0.0176478,0.0302478,-0.0357322,-0.0795366
16.993000000,-0.119508,0.0238622,0.0176274,0.0302443,-0.0357383,-0.0795146
17.027000000,-0.119488,0.0238591,0.017606,0.0302406,-0.0357447,-0.0794915
17.060000000,-0.119467,0.023856,0.0175848,0.030237,-0.035751,-0.0794687
17.093000000,-0.119467,0.023856,0.0175848,0.030237,-0.035751,-0.0794687
17.127000000,-0.119447,0.023853,0.0175638,0.0302334,-0.0357572,-0.079446
17.160000000,-0.119428,0.0238501,0.017543,0.0302299,-0.0357634,-0.0794235
17.193000000,-0.119408,0.0238471,0.0175223,0.0302264,-0.0357696,-0.079401
17.227000000,-0.119388,0.0238442,0.0175017,0.0302229,-0.0357757,-0.0793787
17.260000000,-0.119369,0.0238413,0.0174813,0.0302195,-0.0357818,-0.0793566
17.293000000,-0.119369,0.0238413,0.0174813,0.0302195,-0.0357818,-0.0793566
17.327000000,-0.11935,0.0238385,0.0174611,0.0302161,-0.0357878,-0.0793347
17.360000000,-0.11933,0.0238357,0.017441,0.0302128,-0.0357938,-0.0793129
17.393000000,-0.119311,0.023833,0.017421,0.0302095,-0.0357997,-0.0792912
17.427000000,-0.119292,0.0238303,0.0174012,0.0302062,-0.0358056,-0.0792696
17.460000000,-0.119274,0.0238276,0.0173815,0.030203,-0.0358115,-0.0792482

17.493000000,-0.119255,0.023825,0.017362,0.0301998,-0.0358173,-0.0792269
17.527000000,-0.119255,0.023825,0.017362,0.0301998,-0.0358173,-0.0792269
17.560000000,-0.119237,0.0238224,0.0173426,0.0301967,-0.035823,-0.0792058
17.593000000,-0.119218,0.0238198,0.0173234,0.0301936,-0.0358287,-0.0791848
17.627000000,-0.1192,0.0238173,0.0173042,0.0301905,-0.0358344,-0.0791639
17.660000000,-0.119182,0.0238148,0.0172852,0.0301875,-0.03584,-0.0791432
17.694000000,-0.119164,0.0238123,0.0172664,0.0301845,-0.0358456,-0.0791225
17.727000000,-0.119164,0.0238123,0.0172664,0.0301845,-0.0358456,-0.0791225
17.760000000,-0.119146,0.0238099,0.0172477,0.0301815,-0.0358511,-0.0791021
17.794000000,-0.119128,0.0238075,0.0172287,0.0301785,-0.0358563,-0.0790813
17.827000000,-0.119111,0.0238052,0.0172111,0.0301758,-0.0358609,-0.079062
17.860000000,-0.119094,0.0238029,0.0171927,0.0301729,-0.0358656,-0.0790419
17.894000000,-0.119076,0.0238006,0.0171745,0.0301701,-0.0358704,-0.079022
17.928000000,-0.119076,0.0238006,0.0171745,0.0301701,-0.0358704,-0.079022
17.960000000,-0.119059,0.0237984,0.0171564,0.0301673,-0.035875,-0.0790021
17.993000000,-0.119042,0.0237961,0.0171385,0.0301645,-0.0358796,-0.0789824
18.027000000,-0.119025,0.0237939,0.0171206,0.0301618,-0.0358842,-0.0789628
18.060000000,-0.119008,0.0237918,0.0171029,0.0301591,-0.0358888,-0.0789433
18.093000000,-0.118991,0.0237896,0.0170853,0.0301565,-0.0358933,-0.0789239
18.127000000,-0.118991,0.0237896,0.0170853,0.0301565,-0.0358933,-0.0789239
18.160000000,-0.118975,0.0237875,0.0170678,0.0301538,-0.0358977,-0.0789046
18.193000000,-0.118958,0.0237855,0.0170505,0.0301513,-0.0359021,-0.0788855
18.227000000,-0.118941,0.0237834,0.0170332,0.0301487,-0.0359065,-0.0788665
18.260000000,-0.118925,0.0237814,0.0170161,0.0301462,-0.0359108,-0.0788475
18.293000000,-0.118909,0.0237794,0.016999,0.0301437,-0.035915,-0.0788287
18.327000000,-0.118909,0.0237794,0.016999,0.0301437,-0.035915,-0.0788287
18.360000000,-0.118893,0.0237775,0.0169821,0.0301412,-0.0359192,-0.07881
18.393000000,-0.118877,0.0237755,0.0169654,0.0301388,-0.0359234,-0.0787914
18.427000000,-0.118861,0.0237736,0.0169487,0.0301364,-0.0359276,-0.0787729
18.460000000,-0.118845,0.0237718,0.0169321,0.030134,-0.0359317,-0.0787545
18.493000000,-0.118829,0.0237699,0.0169156,0.0301317,-0.0359357,-0.0787362
18.528000000,-0.118829,0.0237699,0.0169156,0.0301317,-0.0359357,-0.0787362
18.560000000,-0.118813,0.0237681,0.0168992,0.0301293,-0.0359397,-0.078718
18.593000000,-0.118798,0.0237663,0.016883,0.0301271,-0.0359437,-0.0787
18.627000000,-0.118782,0.0237645,0.0168664,0.0301247,-0.0359477,-0.0786815
18.661000000,-0.118766,0.0237627,0.0168496,0.0301224,-0.0359518,-0.0786628
18.693000000,-0.118766,0.0237627,0.0168496,0.0301224,-0.0359518,-0.0786628
18.727000000,-0.11875,0.0237609,0.0168328,0.0301201,-0.0359558,-0.0786441
18.760000000,-0.118735,0.0237593,0.0168178,0.030118,-0.0359595,-0.0786273
18.793000000,-0.11872,0.0237576,0.016802,0.0301159,-0.0359632,-0.0786097
18.827000000,-0.118705,0.023756,0.0167864,0.0301138,-0.035967,-0.0785922
18.860000000,-0.11869,0.0237544,0.0167708,0.0301117,-0.0359707,-0.0785748
18.893000000,-0.11869,0.0237544,0.0167708,0.0301117,-0.0359707,-0.0785748
18.927000000,-0.118675,0.0237528,0.0167554,0.0301096,-0.0359743,-0.0785575
18.960000000,-0.118661,0.0237513,0.01674,0.0301076,-0.035978,-0.0785404
18.993000000,-0.118646,0.0237497,0.0167247,0.0301056,-0.0359816,-0.0785232
19.027000000,-0.118631,0.0237482,0.0167096,0.0301036,-0.0359851,-0.0785062
19.060000000,-0.118617,0.0237467,0.0166945,0.0301016,-0.0359886,-0.0784893
19.093000000,-0.118617,0.0237467,0.0166945,0.0301016,-0.0359886,-0.0784893
19.127000000,-0.118603,0.0237453,0.0166795,0.0300997,-0.0359921,-0.0784724
19.160000000,-0.118588,0.0237438,0.0166647,0.0300978,-0.0359955,-0.0784557
19.193000000,-0.118574,0.0237424,0.0166498,0.0300959,-0.035999,-0.078439
19.227000000,-0.11856,0.023741,0.0166351,0.0300941,-0.0360023,-0.0784224
19.260000000,-0.118546,0.0237397,0.0166205,0.0300922,-0.0360057,-0.078406
19.293000000,-0.118546,0.0237397,0.0166205,0.0300922,-0.0360057,-0.078406
19.327000000,-0.118532,0.0237383,0.016606,0.0300904,-0.036009,-0.0783896
19.360000000,-0.118518,0.023737,0.0165916,0.0300887,-0.0360122,-0.0783733
19.393000000,-0.118504,0.0237357,0.0165772,0.0300869,-0.0360155,-0.0783571
19.427000000,-0.11849,0.0237344,0.0165626,0.0300851,-0.0360188,-0.0783405
19.460000000,-0.118477,0.0237332,0.0165491,0.0300835,-0.0360218,-0.0783252
19.493000000,-0.118477,0.0237332,0.0165491,0.0300835,-0.0360218,-0.0783252
19.527000000,-0.118464,0.023732,0.0165351,0.0300819,-0.0360249,-0.0783093
19.560000000,-0.118449,0.0237306,0.0165197,0.03008,-0.0360283,-0.0782918

19.593000000,-0.118435,0.0237294,0.016505,0.0300783,-0.0360315,-0.0782752
19.627000000,-0.118422,0.0237283,0.0164919,0.0300768,-0.0360344,-0.0782603
19.660000000,-0.118409,0.0237271,0.0164782,0.0300752,-0.0360374,-0.0782447
19.693000000,-0.118409,0.0237271,0.0164782,0.0300752,-0.0360374,-0.0782447
19.727000000,-0.118395,0.023726,0.0164645,0.0300737,-0.0360404,-0.0782291
19.760000000,-0.118382,0.0237249,0.0164509,0.0300721,-0.0360433,-0.0782137
19.793000000,-0.118369,0.0237238,0.0164374,0.0300706,-0.0360463,-0.0781982
19.827000000,-0.118356,0.0237227,0.016424,0.0300691,-0.0360492,-0.0781829
19.860000000,-0.118343,0.0237217,0.0164106,0.0300676,-0.036052,-0.0781676
19.893000000,-0.118343,0.0237217,0.0164106,0.0300676,-0.036052,-0.0781676
19.927000000,-0.118331,0.0237207,0.0163973,0.0300662,-0.0360548,-0.0781525
19.960000000,-0.118318,0.0237197,0.0163842,0.0300648,-0.0360576,-0.0781374
19.993000000,-0.118305,0.0237187,0.016371,0.0300634,-0.0360604,-0.0781223
20.027000000,-0.118292,0.0237177,0.016358,0.030062,-0.0360631,-0.0781074
20.060000000,-0.11828,0.0237168,0.016345,0.0300606,-0.0360659,-0.0780925
20.093000000,-0.11828,0.0237168,0.016345,0.0300606,-0.0360659,-0.0780925
20.127000000,-0.118267,0.0237158,0.0163318,0.0300592,-0.0360686,-0.0780773
20.160000000,-0.118255,0.0237149,0.0163196,0.030058,-0.0360711,-0.0780634
20.193000000,-0.118243,0.023714,0.0163069,0.0300567,-0.0360737,-0.0780487
20.227000000,-0.118231,0.0237132,0.0162942,0.0300554,-0.0360763,-0.0780341
20.260000000,-0.118218,0.0237123,0.0162816,0.0300541,-0.0360789,-0.0780196
20.293000000,-0.118218,0.0237123,0.0162816,0.0300541,-0.0360789,-0.0780196
20.327000000,-0.118206,0.0237115,0.0162691,0.0300529,-0.0360814,-0.0780052
20.360000000,-0.118194,0.0237107,0.0162567,0.0300517,-0.0360839,-0.0779908
20.393000000,-0.118182,0.0237099,0.0162442,0.0300505,-0.0360864,-0.0779765
20.427000000,-0.11817,0.0237091,0.0162319,0.0300493,-0.0360889,-0.0779622
20.460000000,-0.118158,0.0237083,0.0162196,0.0300481,-0.0360914,-0.077948
20.493000000,-0.118158,0.0237083,0.0162196,0.0300481,-0.0360914,-0.077948
20.527000000,-0.118146,0.0237076,0.0162075,0.030047,-0.0360938,-0.0779339
20.560000000,-0.118135,0.0237068,0.0161954,0.0300459,-0.0360963,-0.0779199
20.593000000,-0.118123,0.0237061,0.0161833,0.0300448,-0.0360987,-0.0779059
20.627000000,-0.118111,0.0237054,0.0161713,0.0300437,-0.0361011,-0.077892
20.660000000,-0.1181,0.0237047,0.0161594,0.0300426,-0.0361035,-0.0778781
20.693000000,-0.1181,0.0237047,0.0161594,0.0300426,-0.0361035,-0.0778781
20.727000000,-0.118088,0.0237041,0.0161476,0.0300416,-0.0361059,-0.0778644
20.761000000,-0.118077,0.0237034,0.0161358,0.0300406,-0.0361083,-0.0778507
20.793000000,-0.118065,0.0237028,0.0161241,0.0300396,-0.0361106,-0.077837
20.827000000,-0.118054,0.0237022,0.0161124,0.0300386,-0.036113,-0.0778234
20.860000000,-0.118042,0.0237016,0.0161008,0.0300376,-0.0361153,-0.0778099
20.893000000,-0.118042,0.0237016,0.0161008,0.0300376,-0.0361153,-0.0778099
20.927000000,-0.118031,0.023701,0.0160893,0.0300367,-0.0361177,-0.0777964
20.960000000,-0.11802,0.0237005,0.0160779,0.0300357,-0.03612,-0.077783
20.993000000,-0.118009,0.0236999,0.0160664,0.0300348,-0.0361223,-0.0777696
21.027000000,-0.117998,0.0236994,0.0160551,0.0300339,-0.0361246,-0.0777563
21.060000000,-0.117987,0.0236988,0.0160438,0.030033,-0.0361268,-0.0777431
21.093000000,-0.117987,0.0236988,0.0160438,0.030033,-0.0361268,-0.0777431
21.127000000,-0.117976,0.0236983,0.0160323,0.0300321,-0.0361292,-0.0777296
21.161000000,-0.117965,0.0236979,0.0160217,0.0300313,-0.0361313,-0.0777172
21.193000000,-0.117954,0.0236974,0.0160106,0.0300305,-0.0361335,-0.0777041
21.227000000,-0.117943,0.0236969,0.0159995,0.0300297,-0.0361358,-0.0776911
21.260000000,-0.117933,0.0236965,0.0159885,0.0300289,-0.036138,-0.0776782
21.293000000,-0.117933,0.0236965,0.0159885,0.0300289,-0.036138,-0.0776782
21.327000000,-0.117922,0.0236961,0.0159776,0.0300281,-0.0361402,-0.0776653
21.360000000,-0.117911,0.0236957,0.0159668,0.0300274,-0.0361424,-0.0776525
21.393000000,-0.117901,0.0236953,0.0159559,0.0300266,-0.0361446,-0.0776397
21.427000000,-0.11789,0.0236949,0.0159452,0.0300259,-0.0361467,-0.077627
21.460000000,-0.11788,0.0236945,0.0159345,0.0300252,-0.0361489,-0.0776143
21.493000000,-0.11788,0.0236945,0.0159345,0.0300252,-0.0361489,-0.0776143
21.527000000,-0.117869,0.0236942,0.0159239,0.0300245,-0.0361511,-0.0776017
21.560000000,-0.117859,0.0236938,0.0159133,0.0300238,-0.0361532,-0.0775892
21.593000000,-0.117848,0.0236935,0.0159027,0.0300231,-0.0361553,-0.0775767
21.627000000,-0.117838,0.0236932,0.0158922,0.0300225,-0.0361574,-0.0775642
21.660000000,-0.117828,0.0236929,0.0158818,0.0300218,-0.0361596,-0.0775518

21.693000000,-0.117828,0.0236929,0.0158818,0.0300218,-0.0361596,-0.0775518
21.727000000,-0.117818,0.0236926,0.0158714,0.0300212,-0.0361617,-0.0775395
21.760000000,-0.117807,0.0236923,0.0158611,0.0300206,-0.0361637,-0.0775272
21.793000000,-0.117797,0.0236921,0.0158508,0.03002,-0.0361658,-0.077515
21.827000000,-0.117787,0.0236918,0.0158406,0.0300194,-0.0361679,-0.0775028
21.860000000,-0.117777,0.0236916,0.0158304,0.0300189,-0.03617,-0.0774907
21.893000000,-0.117777,0.0236916,0.0158304,0.0300189,-0.03617,-0.0774907
21.927000000,-0.117767,0.0236914,0.0158203,0.0300183,-0.036172,-0.0774786
21.960000000,-0.117757,0.0236912,0.01581,0.0300178,-0.0361741,-0.0774663
21.994000000,-0.117748,0.023691,0.0158005,0.0300173,-0.036176,-0.0774549
22.028000000,-0.117738,0.0236908,0.0157906,0.0300168,-0.036178,-0.077443
22.060000000,-0.117728,0.0236906,0.0157807,0.0300163,-0.03618,-0.0774311
22.093000000,-0.117728,0.0236906,0.0157807,0.0300163,-0.03618,-0.0774311
22.127000000,-0.117718,0.0236905,0.0157708,0.0300159,-0.036182,-0.0774193
22.160000000,-0.117709,0.0236904,0.015761,0.0300154,-0.036184,-0.0774075
22.193000000,-0.117699,0.0236902,0.0157512,0.030015,-0.036186,-0.0773958
22.227000000,-0.11769,0.0236901,0.0157415,0.0300145,-0.036188,-0.0773841
22.260000000,-0.11768,0.02369,0.0157319,0.0300141,-0.0361899,-0.0773725
22.293000000,-0.11768,0.02369,0.0157319,0.0300141,-0.0361899,-0.0773725
22.327000000,-0.11767,0.0236899,0.0157223,0.0300137,-0.0361919,-0.0773609
22.360000000,-0.117661,0.0236898,0.0157127,0.0300133,-0.0361938,-0.0773494
22.393000000,-0.117652,0.0236898,0.0157032,0.030013,-0.0361958,-0.0773379
22.427000000,-0.117642,0.0236897,0.0156937,0.0300126,-0.0361977,-0.0773264
22.460000000,-0.117633,0.0236897,0.0156843,0.0300123,-0.0361996,-0.0773151
22.493000000,-0.117633,0.0236897,0.0156843,0.0300123,-0.0361996,-0.0773151
22.527000000,-0.117624,0.0236897,0.0156749,0.0300119,-0.0362015,-0.0773037
22.560000000,-0.117614,0.0236896,0.0156655,0.0300116,-0.0362034,-0.0772924
22.593000000,-0.117605,0.0236896,0.015656,0.0300113,-0.0362054,-0.0772809
22.627000000,-0.117596,0.0236896,0.0156472,0.030011,-0.0362072,-0.0772702
22.660000000,-0.117587,0.0236897,0.015638,0.0300107,-0.036209,-0.077259
22.693000000,-0.117587,0.0236897,0.015638,0.0300107,-0.036209,-0.077259
22.727000000,-0.117578,0.0236897,0.0156289,0.0300105,-0.0362109,-0.0772479
22.760000000,-0.117569,0.0236897,0.0156198,0.0300102,-0.0362127,-0.0772368
22.793000000,-0.11756,0.0236898,0.0156107,0.03001,-0.0362146,-0.0772258
22.827000000,-0.117551,0.0236899,0.0156016,0.0300097,-0.0362164,-0.0772148
22.861000000,-0.117542,0.0236899,0.0155927,0.0300095,-0.0362183,-0.0772038
22.893000000,-0.117542,0.0236899,0.0155927,0.0300095,-0.0362183,-0.0772038
22.927000000,-0.117533,0.02369,0.0155837,0.0300093,-0.0362201,-0.0771929
22.960000000,-0.117524,0.0236901,0.0155748,0.0300091,-0.0362219,-0.0771821
22.993000000,-0.117515,0.0236902,0.015566,0.030009,-0.0362237,-0.0771713
23.027000000,-0.117506,0.0236904,0.0155563,0.0300088,-0.0362257,-0.0771594
23.060000000,-0.117497,0.0236905,0.0155473,0.0300086,-0.0362275,-0.0771484
23.093000000,-0.117497,0.0236905,0.0155471,0.0300086,-0.0362276,-0.0771481
23.127000000,-0.117488,0.0236906,0.0155388,0.0300085,-0.0362293,-0.077138
23.160000000,-0.11748,0.0236908,0.0155301,0.0300084,-0.036231,-0.0771274
23.193000000,-0.117471,0.023691,0.0155215,0.0300083,-0.0362328,-0.0771168
23.227000000,-0.117462,0.0236911,0.0155129,0.0300081,-0.0362346,-0.0771062
23.260000000,-0.117454,0.0236913,0.0155043,0.0300081,-0.0362363,-0.0770956
23.293000000,-0.117454,0.0236913,0.0155043,0.0300081,-0.0362363,-0.0770956
23.327000000,-0.117445,0.0236915,0.0154956,0.030008,-0.0362381,-0.0770849
23.360000000,-0.117437,0.0236917,0.0154876,0.0300079,-0.0362397,-0.077075
23.393000000,-0.117429,0.0236919,0.0154791,0.0300079,-0.0362415,-0.0770646
23.427000000,-0.11742,0.0236922,0.0154707,0.0300078,-0.0362432,-0.0770542
23.460000000,-0.117412,0.0236924,0.0154623,0.0300078,-0.0362449,-0.0770439
23.493000000,-0.117412,0.0236924,0.0154623,0.0300078,-0.0362449,-0.0770439
23.527000000,-0.117404,0.0236927,0.015454,0.0300078,-0.0362466,-0.0770336
23.560000000,-0.117395,0.0236929,0.0154458,0.0300078,-0.0362483,-0.0770234
23.593000000,-0.117387,0.0236932,0.0154375,0.0300077,-0.03625,-0.0770132
23.627000000,-0.117379,0.0236935,0.0154293,0.0300078,-0.0362517,-0.077003
23.660000000,-0.117371,0.0236938,0.0154211,0.0300078,-0.0362534,-0.0769929
23.693000000,-0.117371,0.0236938,0.0154211,0.0300078,-0.0362534,-0.0769929
23.727000000,-0.117363,0.0236941,0.015413,0.0300078,-0.036255,-0.0769828
23.760000000,-0.117354,0.0236944,0.0154049,0.0300079,-0.0362567,-0.0769727

```
23.793000000,-0.117346,0.0236947,0.0153968,0.0300079,-0.0362583,-0.0769627
23.827000000,-0.117338,0.023695,0.0153888,0.030008,-0.03626,-0.0769527
23.860000000,-0.11733,0.0236953,0.0153808,0.0300081,-0.0362616,-0.0769428
```

Example of log.txt file

The file contains the starting and ending time for experiments listed by their filename.

```
/home/jdurand7/Dropbox/phd/code/modeling/microscopic/launch/robotarium_2_0.01.launch
1479772657.445741513
1479772691.545704047
/home/jdurand7/Dropbox/phd/code/modeling/microscopic/launch/robotarium_2_0.02.launch
1479772691.546762821
1479772736.973055900
/home/jdurand7/Dropbox/phd/code/modeling/microscopic/launch/robotarium_2_0.03.launch
1479772736.975268721
1479772763.430581048
/home/jdurand7/Dropbox/phd/code/modeling/microscopic/launch/robotarium_2_0.04.launch
1479772763.431503844
1479772786.057706638
/home/jdurand7/Dropbox/phd/code/modeling/microscopic/launch/robotarium_2_0.05.launch
1479772786.058781728
1479772808.015162025
/home/jdurand7/Dropbox/phd/code/modeling/microscopic/launch/robotarium_2_0.06.launch
1479772808.016130592
1479772834.555382629
/home/jdurand7/Dropbox/phd/code/modeling/microscopic/launch/robotarium_2_0.07.launch
1479772834.556152737
1479772856.892333712
/home/jdurand7/Dropbox/phd/code/modeling/microscopic/launch/robotarium_2_0.08.launch
1479772856.894710700
1479772878.288055537
/home/jdurand7/Dropbox/phd/code/modeling/microscopic/launch/robotarium_2_0.09.launch
1479772878.288974535
1479772904.205606228
/home/jdurand7/Dropbox/phd/code/modeling/microscopic/launch/robotarium_2_0.10.launch
1479772904.206371317
1479772926.293962170
/home/jdurand7/Dropbox/phd/code/modeling/microscopic/launch/robotarium_3_0.01.launch
1479772926.294981850
1479772986.088835384
/home/jdurand7/Dropbox/phd/code/modeling/microscopic/launch/robotarium_3_0.02.launch
1479772986.089844818
1479773012.056035298
/home/jdurand7/Dropbox/phd/code/modeling/microscopic/launch/robotarium_3_0.03.launch
1479773012.056896150
1479773049.524392870
/home/jdurand7/Dropbox/phd/code/modeling/microscopic/launch/robotarium_3_0.04.launch
1479773049.525498153
```

B.1.4 Real system

This section contains all the files necessary to setup the required 40 experiments and submit them online to the Robotarium interface.

generateMainFiles.m

Generates a set of 40 *main.m* files which can be submitted to the Robotarium platform. The main file is the one run on Robotarium and it calls an additional *experiment.m* file which must be submitted as well (see below). Note that this script uses a configuration file of initial poses. Such a file can be generated thanks to the script *generateInitialPositions.m* (see next subsection).

```
% Clean and prepare workspace
clc
close all
clear

% Define experiments range and generate full factorial DOE
Nvec = 2:5;
Vvec = .01:0.01:0.1;
[N,V] = meshgrid(Nvec,Vvec);

fprintf('Creating %d experiments...', numel(N))

% Generate files
for i = 1:size(N,1)
    for j = 1:size(N,2)
        % Prepare string for initial conditions
        %     x0 = generate_initial_conditions(N(i,j));

load(sprintf('../initialPoses/init_poses_%d_%.2f.mat',N(i,j),V(i,j)))
    x0_str = strcat('[',...
        sprintf('%s,',x0(1,:)),...
        ';',...
        sprintf('%s,',x0(2,:)),...
        ';',...
        sprintf('%s,',x0(3,:)),...
        ']');

        % Write into file
        fid =
fopen(sprintf('experiments/main %d %d.m',N(i,j),round(100*V(i,j))), 'w')
;
        fprintf(fid, 'experiment (%d,%.2f,%s);\n',...
            N(i,j),...
            V(i,j),...
            x0_str);
        fclose(fid);
    end
end
```



```
fprintf('[OK]\nCheck experiments folder.\n')
```

Example of generated main file: main_3_2.m

```
experiment(3,0.02,[1.000000e-01,-2.000000e-01,1.110223e-16,;5.000000e-02,5.000000e-02,-1.500000e-01,;1.533965e+00,2.080974e+00,5.550957e+00,]);
```

experiment.m

Very similar to *rendezVousMesoscopic.m* for the exception of the Robotarium builders, this file parks the robots at their supposed initial location with the appropriate heading, and then goes on to perform a static consensus. It can be run either in the Robotarium simulator or on the real system (through the previously mentioned main files).

```
function experiment(N,v,initial_conditions)
% Experiment
% 1 - Initializes N robots at initial_conditions
% 2 - Performs static consensus with a maximum linear speed saturated
at v
%
% Jean-Guillaume Durand
% jdurand7@gatech.edu
% 2016

%% 1 - Initialize N robots at initial conditions
% Get Robotarium object used to communicate with the robots/simulator
rb = RobotariumBuilder();

% Get the number of available agents from the Robotarium. We don't
need a
% specific value for this algorithm
N_available = rb.get_available_agents();

% If not enough robots for experiment, stop
if N_available < N, return, end

% Set the number of agents and whether we would like to save data.
Then,
% build the Robotarium simulator object!
r = rb.set_number_of_agents(N).set_save_data(false).build();

% Initialize x so that we don't run into problems later. This isn't
always
% necessary
```

```

x = r.get_poses();
r.step();

% Set some parameters for use with the barrier certificates. We don't
want
% our agents to collide
safety = 0.05;
lambda = 0.01;

% Create a barrier certificate for use with the above parameters
unicycle_barrier_certificate =
create_uni_barrier_certificate('SafetyRadius', safety, ...
    'ProjectionDistance', lambda);

% Create parking controller
args = {'PositionError', 0.01, 'RotationError', 0.25};
init_checker = create_is_initialized(args{:});
automatic_parker = create_automatic_parking_controller(args{:});

while(~init_checker(x, initial_conditions))
    % Compute velocities
    x = r.get_poses();
    dxu = automatic_parker(x, initial_conditions);
    dxu = unicycle_barrier_certificate(dxu, x);
    % Update
    r.set_velocities(1:N, dxu);
    r.step();
end

%% 2 - Perform static consensus with a maximum linear speed saturated
at v
% Experiment constants
% Generate a cyclic graph Laplacian from our handy utilities. For this
% algorithm, any connected graph will yield consensus
L = cycleGL(N);

% Grab tools we need to convert from single-integrator to unicycle
dynamics
% Gain for the diffeomorphism transformation between single-integrator
and
% unicycle dynamics
[si_to_uni_dyn, uni_to_si_states] =
create_si_to_uni_mapping('ProjectionDistance', lambda);

si_barrier_cert = create_si_barrier_certificate('SafetyRadius',
safety);

% Select the number of iterations for the experiment. This value is
% arbitrary
iterations = 5000; % Maximum time at around 3 minutes

% Initialize velocity vector for agents. Each agent expects a 2 x 1
% velocity vector containing the linear and angular velocity,
respectively.
dxi = zeros(2, N);

```

```

%Iterate for the previously specified number of iterations
for t = 1:iterations
    % Retrieve the most recent poses from the Robotarium. The time
    delay is
    % approximately 0.033 seconds
    x = r.get_poses();

    % Convert to SI states
    xi = uni_to_si_states(x);

    % Algorithm
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for i = 1:N
        % Initialize velocity to zero for each agent. This allows us
to sum
        %over agent i's neighbors
        dxi(:, i) = [0 ; 0];

        % Get the topological neighbors of agent i based on the graph
        %Laplacian L
        neighbors = topological_neighbors(L, i);

        % Iterate through agent i's neighbors
        for j = neighbors

            % For each neighbor, calculate appropriate consensus term
and
            %add it to the total velocity
            dxi(:, i) = dxi(:, i) + (xi(:, j) - xi(:, i));
        end
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    % Utilize barrier certificates
    dxi = si_barrier_cert(dxi, xi);

    % Threshold here instead

    norms = arrayfun(@(i) norm(dxi(:, i)), 1:N);
    need_to_be_thresh = find(norms > v);
    dxi(:, need_to_be_thresh) = cell2mat(arrayfun(@(i) v*dxi(:, i) /
norms(i), need_to_be_thresh, 'UniformOutput', false));

    % Transform the single-integrator to unicycle dynamics using the
the
    % transformation we created earlier
    dxu = si_to_uni_dyn(dxi, x);

    % Impose velocity v on agents
    % linear = dxu(1,:);
    % linear(linear > v) = v;
    % linear(linear < -v) = -v;
    % dxu(1,:) = linear;

```

```

    % Set velocities of agents 1,...,N
    r.set_velocities(1:N, dxu);

    % Send the previously set velocities to the agents. This function
    must be called!
    r.step();
end

% Though we didn't save any data, we still should call
r.call_at_scripts_end() after our
% experiment is over!
r.call_at_scripts_end();
end

```

The results from the Robotarium were analyzed in two different ways: one with a script to compute the consensus location, the other by hand by looking at the experiment videos to correctly compute the consensus time. The following scripts helped in analyzing the results.

analyzeRobotariumData.m

Script which automatically computes the consensus time and the consensus location based on the Robotarium output files.

```

% Prepare workspace
clc
close all
clear

N = 3;
v = 0.08;

%% Load files
foldername = sprintf('results/%d_%.2f/',N,v);
files = dir(fullfile(foldername, '*.mat'));
filename = files(end).name; % Take the last file
load(sprintf('%s/%s', foldername, filename))
load(sprintf('../initialPoses/init_poses_%.2f.mat',N,v))

x = robotarium_data(1:5:end,:);
y = robotarium_data(2:5:end,:);
N = size(x,1);

```

```

% Compute time to reach consensus
timestep = 1/30;
is = 600;
i = is;
d = 0;
d_prev = 1;
count = 0;
while i < size(x,2) && count < 25
    if abs(d - d_prev) < 1e-5
        count = count + 1;
    end
    d_prev = d;
    dx = x(:,i) - x(:,i-1);
    dy = y(:,i) - y(:,i-1);
    d = mean(sqrt(dx.^2 + dy.^2));
    % Increment
    i = i + 1;
end
ic = i;
tc = (ic-is)*timestep;

% Compute consensus location
xc = mean(x(:,end));
yc = mean(y(:,end));

y0 = x0(2,:)';
x0 = x0(1,:)';

%% Plot
figure
hold on
plot(xc,yc,'xk')
plot(x(:,1)',y(:,1)', 'or') % Start point
plot(x(:,is)',y(:,is)', 'om') % Start of consensus time
plot(x(:,end)',y(:,end)', 'og') % End point
plot(x',y', '-')
plot(x0,y0, '+k')
hold off
xlabel('x (m)', 'FontName', 'Times New Roman', 'FontSize', 12)
ylabel('y (m)', 'FontName', 'Times New Roman', 'FontSize', 12)

legend_struct = cell(1,N+3);
legend_struct{1} = 'Initial conditions barycenter';
legend_struct{2} = 'Random initial locations';
legend_struct{3} = 'Starting locations';
legend_struct{4} = 'Consensus locations';
for i = 1:N
    legend_struct{i+4} = sprintf('Gritbot %d',i);
end
h = legend(legend_struct);
set(h, 'FontName', 'Times New Roman', 'FontSize', 12)

set(gca, 'FontName', 'Times New Roman', 'FontSize', 12)
axis equal
axis([-0.6 0.6 -0.35 0.35])

```

```

% Display
fprintf('N = %d, v = %.2fcm/s\n',N,v)
fprintf('Time = %fs\n',tc)
fprintf('Position = [%f, %f]\n',xc,yc)

```

generateMatFile.m

Once the results have been properly reported in a *real.csv* file, this script reformats the data into a proper MAT-file using the same format used for the other modeling techniques.

```

% Prepare workspace
clc
close all
clear

% Load data
data = csvread('real.csv');
t = data(:,1);
T = data(:,2);
x = data(:,3);
y = data(:,4);

% Format the data
t_real = reshape(t,10,4);
T_real = reshape(T,10,4);
x = reshape(x,10,4);
y = reshape(y,10,4);

X_real = cell(10,4);
for i = 1:size(X_real,1)
    for j = 1:size(X_real,2)
        X_real{i,j} = [x(i,j);y(i,j)];
    end
end

% Save formatted data
save real.mat

```

real.csv

Contains the results of the Robotarium experimentation, the first column is the experiment time, the second is the consensus time while the last two columns are the x and y

coordinates of the consensus location. The previous file *generateMatFile.m* explains how to associate each row here with a set of inputs N and v .

```
102,29.233333,0.205051,-0.046544
102,20.2,-0.142806,0.091981
102,16,0.20383,-0.09421
102,10.8,-0.033532,-0.146873
102,12.733333,0.054997,-0.062684
102,13.633333,-0.100632,0.015175
102,22.433333,0.289793,0.19845
102,8.6,-0.17332,0.000062
102,15.766667,0.321387,0.261598
101,8.466667,0.352052,-0.000506
102,48.3,-0.067764,0.016682
102,10.033333,-0.049521,-0.007815
102,32.1,0.080985,0.038893
102,8.5,0.3133,0.0509
101,26.233333,0.075515,-0.010795
102,10.2,-0.039294,-0.078337
102,9.566667,0.161283,0.177629
80,35.166667,-0.338886,0.027406
102,8.166667,0.201344,-0.074088
101,10.633333,-0.00741,0.013563
102,47.5,-0.089437,-0.029958
102,39.1,0.007485,-0.001016
101,28.7,0.163564,-0.013037
102,27.4,0.0197,0.0856
102,34.766667,-0.030778,-0.121071
102,18.766667,-0.098128,0.062327
98,26.133333,-0.181069,-0.127278
101,10.23,-0.089852,-0.050806
81,17.233333,-0.027068,0.008907
91,28.366667,-0.173243,-0.058091
102,68.9,0.171482,0.038068
101,35.3,0.027043,-0.046788
102,32.066667,0.133503,0.063558
102,30.233333,-0.139308,-0.031819
102,37.733333,-0.132491,0.146308
102,24.7,-0.192258,0.094753
102,22.1,-0.10701,-0.009696
77,16.166667,-0.068914,-0.179585
102,34.833333,0.005551,-0.047264
102,29.566667,0.009603,0.01186
```

B.1.5 Scripts

generateInitialPositions.m

A script generating random initial poses for the robots based on a given design of experiments. The same set of initial positions was used to compare each type of modeling technique.

```

% Prepare workspace
clc
close all
clear

%% Design space
Nvec = 2:5;
Vvec = .01:0.01:0.1;
safetyRadius = 0.1; % m

[N,V] = meshgrid(Nvec,Vvec);

%% Analysis
n1 = size(N,1);
n2 = size(N,2);
for i = 1:n1
    for j = 1:n2
        fprintf('%d/%d (%.2f%%)\n', (i-1)*n2 + j, n1*n2, 100*((i-1)*n2 +
j)/(n1*n2))
        % Generate initial positions
        x0 = zeros(3,N(i,j));
        numX = floor(1.2 / safetyRadius);
        numY = floor(0.7 / safetyRadius);
        values = randperm(numX * numY, N(i,j));
        for k = 1:N(i,j)
            [x, y] = ind2sub([numX numY], values(k));
            x = x*safetyRadius - 0.6;
            y = y*safetyRadius - 0.35;
            x0(1:2,k) = [x, y]';
        end
        x0(3,:) = rand(1, N(i,j))*2*pi;

        % Save
        save(sprintf('init_poses_%d_%.2f.mat',N(i,j),V(i,j)), 'x0')
    end
end
end

```

plot_models.m

The script used to generate the different figures of the modeling section.

```

% Prepare workspace
clc
close all
clear

```



```

colors = get(groot, 'DefaultAxesColorOrder');

% Load data
load('macroscopic/macroscopic.mat')
load('mesoscopic/mesoscopic.mat')
load('microscopic/microscopic.mat')
load('real/real.mat')

%% Direct responses
figure
hold on
surf(N,V,T_macro, 'FaceColor', colors(1,:))
surf(N,V,T_meso, 'FaceColor', colors(2,:))
surf(N,V,T_micro, 'FaceColor', colors(3,:))
surf(N,V,T_real, 'FaceColor', colors(4,:))
hold off
xlabel('N', 'FontName', 'Times New Roman', 'FontSize', 12)
ylabel('v (m/s)', 'FontName', 'Times New Roman', 'FontSize', 12)
zlabel('Consensus time (s)', 'FontName', 'Times New Roman', 'FontSize', 12)
h = legend('Macroscopic', 'Mesoscopic', 'Microscopic', 'Real system');
set(h, 'FontName', 'Times New Roman', 'FontSize', 12)
set(gca, 'FontName', 'Times New Roman', 'FontSize', 12)

figure
hold on
surf(N,V,t_macro, 'FaceColor', colors(1,:))
surf(N,V,t_meso, 'FaceColor', colors(2,:))
surf(N,V,t_micro, 'FaceColor', colors(3,:))
surf(N,V,t_real, 'FaceColor', colors(4,:))
hold off
xlabel('N', 'FontName', 'Times New Roman', 'FontSize', 12)
ylabel('v (m/s)', 'FontName', 'Times New Roman', 'FontSize', 12)
zlabel('Execution time (s)', 'FontName', 'Times New Roman', 'FontSize', 12)
h = legend('Macroscopic', 'Mesoscopic', 'Microscopic', 'Real system');
set(h, 'FontName', 'Times New Roman', 'FontSize', 12)
set(gca, 'FontName', 'Times New Roman', 'FontSize', 12)

%% Section plots
i = 1;

V(i,1)

figure
hold on
plot(N(i,:), T_macro(i,:), 'Color', colors(1,:))
plot(N(i,:), T_meso(i,:), 'Color', colors(2,:))
plot(N(i,:), T_micro(i,:), 'Color', colors(3,:))
plot(N(i,:), T_real(i,:), 'Color', colors(4,:))
hold off
xlabel('N', 'FontName', 'Times New Roman', 'FontSize', 12)
ylabel('Consensus time (s)', 'FontName', 'Times New Roman', 'FontSize', 12)
h = legend('Macroscopic', 'Mesoscopic', 'Microscopic', 'Real system');
set(h, 'FontName', 'Times New Roman', 'FontSize', 12)
set(gca, 'FontName', 'Times New Roman', 'FontSize', 12)

```

```

%% Error on consensus time
err_t_macro = reshape(abs(T_macro - T_real)./T_real,[],1);
err_t_meso = reshape(abs(T_meso - T_real)./T_real,[],1);
err_t_micro = reshape(abs(T_micro - T_real)./T_real,[],1);

mean_err_t_macro = 100*mean(err_t_macro);
mean_err_t_meso = 100*mean(err_t_meso);
mean_err_t_micro = 100*mean(err_t_micro);

figure
hold on
bar(1,mean_err_t_macro,'FaceColor',colors(1,:))
bar(2,mean_err_t_meso,'FaceColor',colors(2,:))
bar(3,mean_err_t_micro,'FaceColor',colors(3,:))
hold off
ylabel('Percentage error on consensus time (%)','FontName','Times New Roman','FontSize',12)
set(gca,'FontName','Times New Roman','FontSize',12)
set(gca,'XTick',[1 2 3])
set(gca,'XTickLabels',{'Macroscopic','Mesoscopic','Microscopic'})

%% Absolute error on consensus position
ERR_x_macro = zeros(size(X_macro));
ERR_x_meso = zeros(size(X_macro));
ERR_x_micro = zeros(size(X_macro));
for i = 1:size(X_real,1)
    for j = 1:size(X_real,2)
        ERR_x_macro(i,j) = norm(X_macro{i,j} - X_real{i,j});
        ERR_x_meso(i,j) = norm(X_meso{i,j} - X_real{i,j});
        ERR_x_micro(i,j) = norm(X_micro{i,j} - X_real{i,j});
    end
end

err_x_macro = mean(mean(ERR_x_macro));
err_x_meso = mean(mean(ERR_x_meso));
err_x_micro = mean(mean(ERR_x_micro));

figure
hold on
bar(1,err_x_macro*100,'FaceColor',colors(1,:))
bar(2,err_x_meso*100,'FaceColor',colors(2,:))
bar(3,err_x_micro*100,'FaceColor',colors(3,:))
hold off
ylabel('Absolute error on consensus position (cm)','FontName','Times New Roman','FontSize',12)
set(gca,'FontName','Times New Roman','FontSize',12)
set(gca,'XTick',[1 2 3])
set(gca,'XTickLabels',{'Macroscopic','Mesoscopic','Microscopic'})

%% Execution time
mean_t_macro = mean(mean(t_macro));
mean_t_meso = mean(mean(t_meso));
data = csvread('microscopic/logs/runtimes.txt');
t_micro = data(:,2) - data(:,1);

```

```

mean_t_micro = mean(mean(t_micro));
mean_t_real = mean(mean(t_real));

figure
hold on
bar(1,mean_t_macro,'FaceColor',colors(1,:))
bar(2,mean_t_meso,'FaceColor',colors(2,:))
bar(3,mean_t_micro,'FaceColor',colors(3,:))
bar(4,mean_t_real,'FaceColor',colors(4,:))
hold off
ylabel('Average execution time (s)','FontName','Times New
Roman','FontSize',12)
set(gca,'FontName','Times New Roman','FontSize',12)
set(gca,'XTick',[1 2 3 4])
set(gca,'XTickLabels',{'Macroscopic','Mesoscopic','Microscopic','Real
system'})
set(gca,'YScale','log');

```

B.2 Optimization

The code used to show the advantage of simultaneous optimization and the benefits of the proposed optimization scheme.

B.2.1 Sequential and simultaneous optimization

The files used to analyze the canonical mapping mission with respect to different optimization schemes.

main.m

The main file trying to optimize a swarm configuration based on the settings and properties of the mapping problem.

```

% Numerality vs. Individual Performance Tradeoff
% Jean-Guillaume Durand (jean-guillaume.durand@gatech.edu)
% Aerospace Systems Design Laboratory (ASDL)
% Georgia Institute of Technology
% 2016

% Homogeneous group optimization

% Clean the workspace
clc

```

```

close all
clear variables

addpath('utilities')

%% Problem parameters
% Environment -----
---
problem.environment.d0 = 100; % Distance from base to mission
problem.environment.lx = 100; % Map horizontal size
problem.environment.ly = 100; % Map vertical size

% Cost variables
problem.cost.C0 = 10; % Swarm fixed cost (ground station...)
problem.cost.c0 = 3; % Agent fixed cost
problem.cost.cv = 1; % Cost of one unit of speed
problem.cost.cv2 = 0;

% problem.cost.C0 = 10; % Swarm fixed cost (ground station...)
% problem.cost.c0 = 3; % Agent fixed cost
% problem.cost.cv = 1; % Cost of one unit of speed
% problem.cost.cv2 = 0.3;

% Constraint
problem.cost.Cmax = 70; % Maximum cost
% problem.cost.Cmax = 100; % Maximum cost

% Number of architectures
problem.nArchis = 1;

%% Run mission
% Experiment 1
x_mM = [4,9.999999993264135];
x_Mm = [7,5.571459922656952];
x_s = [5,9.000199990881889];
f_mM = 2.850000000616806e+02;
f_Mm = 3.230717177359792e+02;
f_s = 2.622163959912607e+02;
c_mM = 61.999999991343074;
c_Mm = 70.000628369138080;
c_s = 70.000999862698440;

% Experiment 2
% x_mM = [5,7.000938611068547];
% x_Mm = [7,5.571459922656952];
% x_s = [5,9.000199990881889];
% f_mM = 3.370976566297580e+02;
% f_Mm = 3.230717177359792e+02;
% f_s = 2.622163959912607e+02;
% c_mM = 60.004693055342734;
% c_Mm = 70.000628369138080;
% c_s = 70.000999862698440;

% Experiment 3 (quadratic cost constraint at 100)
% x_mM = [2,9.999999994786457];

```

```

% x_mM = [5,5.598186044422418];
% x_s = [7,4.302872355432176];
% f_mM = 5.300000002763178e+02;
% f_Mm = 4.215651250732037e+02;
% f_s = 4.183252142554459e+02;
% c_mM = 95.999999927010390;
% c_Mm = 1.000004607040610e+02;
% c_s = 1.000009985530244e+02;

% Experiment 4
% x_mM = [8,3.443486685868475];
% x_Mm = [5,5.598176762543418];
% x_s = [6,4.873848085169360];
% f_mM = 4.719054110674343e+02;
% f_Mm = 4.215658240358566e+02;
% f_s = 4.171925956249162e+02;
% c_mM = 90.006134820756090;
% c_Mm = 1.000002584097388e+02;
% c_s = 1.000009997941724e+02;

% Experiment 5
% x_mM = [5,9.000199990881889];
% x_Mm = [7,5.571459922656952];
% x_s = [5,9.000199990881889];
% f_mM = 2.622163959912607e+02;
% f_Mm = 3.230717177359792e+02;
% f_s = 2.622163959912607e+02;
% c_mM = 70.000999862698440;
% c_Mm = 70.000628369138080;
% c_s = 70.000999862698440;

% Experiment 6
% x_mM = [5,5.598190765561446];
% x_Mm = [5,5.598207437216443];
% x_s = [6,4.873848080260603];
% f_mM = 4.215647695534209e+02;
% f_Mm = 4.215635141189849e+02;
% f_s = 4.171925960450970e+02;
% c_mM = 1.000005635992334e+02;
% c_Mm = 1.000009269512405e+02;
% c_s = 1.000009996785916e+02;

% tic
% [x_mM,f_mM,c_mM] = micro_Macro_optimizer(problem);
% [x_Mm,f_Mm,c_Mm] = Macro_micro_optimizer(problem);
% [x_s,f_s,c_s] = simultaneous_optimizer(problem);
% toc

%% Plot
f = @(x)myfitnessfcn(x,problem.environment);
c = @(x)(mynonlcon(x, problem.cost) + problem.cost.Cmax);
Nmax = 10;
vmax = 11;
fontSize = 12;

[N, v] = meshgrid(1:1:Nmax, 1:1:vmax);

```

```

T = zeros(size(N));
C = zeros(size(N));

for i = 1:size(N,1)
    for j = 1:size(N,2);
        T(i,j) = f([N(i,j),v(i,j)]);
        C(i,j) = c([N(i,j),v(i,j)]);
    end
end

Tc = T;
Tc(C <= problem.cost.Cmax) = 0;

%% Plot 1
figure
surf(N,v,T-1, 'EdgeColor', 'none')
hold on
% Constraint
hc = surf(N,v,Tc, 'EdgeColor', 'none', 'FaceColor', 'r', 'FaceAlpha', 0.5);
% Points
h_mM =
plot3(x_mM(1),x_mM(2),f([x_mM(1),x_mM(2)]), 'o', 'MarkerEdgeColor', 'k', 'MarkerFaceColor', 'c');
h_Mm =
plot3(x_Mm(1),x_Mm(2),f([x_Mm(1),x_Mm(2)]), 'o', 'MarkerEdgeColor', 'k', 'MarkerFaceColor', 'm');
hx =
plot3(x_s(1),x_s(2),f([x_s(1),x_s(2)]), 'o', 'MarkerEdgeColor', 'k', 'MarkerFaceColor', 'r');
% Lines
N1 = zeros(size(N(1,:))); vl = zeros(size(N(1,:))); l =
zeros(size(N(1,:)));
for i = 1:size(N,2)
    N1(i) = N(1,i); vl(i) = x_mM(2); l(i) = f([N1(i),vl(i)]);
end
plot3(N1,vl,l, 'c--')

N1 = zeros(size(N(1,:))); vl = zeros(size(N(1,:))); l =
zeros(size(N(1,:)));
for i = 1:size(N,1)
    N1(i) = x_Mm(1); vl(i) = v(i,1); l(i) = f([N1(i),vl(i)]);
end
plot3(N1,vl,l, 'm--')
hold off

xlabel('N', 'FontSize', fontSize, 'FontName', 'Times New Roman')
ylabel('v (m/s)', 'FontSize', fontSize, 'FontName', 'Times New Roman')
zlabel('Mapping time (s)', 'FontSize', fontSize, 'FontName', 'Times New Roman')
h = legend([hc,h_mM,h_Mm,hx], {'Constraint', 'Micro-macro optimum', 'Macro-micro optimum', 'Global optimum'}, 'location', 'northeast');
set(h, 'FontSize', fontSize, 'FontName', 'Times New Roman')
set(gca, 'FontSize', fontSize, 'FontName', 'Times New Roman')
hc = colorbar;

```

```

% shading interp

set(gca, 'ZScale', 'log')
set(gca, 'ZLim', [100 10^4])
% set(gcf, 'Position', [1 1 1252 750])
view(115,16)

% Transparent background
set(gcf, 'Color', 'None')
set(gca, 'Color', 'None', 'XColor', 'w', 'YColor', 'w', 'ZColor', 'w')
set(h, 'color', 'none', 'TextColor', 'w', 'EdgeColor', 'w')
set(hc, 'color', 'w');

%% Plot 2
% PCA
X = [reshape(N,[],1), reshape(v,[],1), reshape(T,[],1)];
coeff = pca(X);
x0 = [x_s(1) x_s(2) f_s]';
x01 = [x0, x0 + 250*coeff(:,1)];
x02 = [x0, x0 + 1*coeff(:,2)];
x03 = [x0, x0 + 1*coeff(:,3)];

figure
hold on
surf(N,v,T-1, 'EdgeColor', 'none')%, 'FaceAlpha', 0.5)
plot3(x01(1,:), x01(2,:), x01(3,:), 'r', 'LineWidth', 2)
plot3(x02(1,:), x02(2,:), x02(3,:), 'g', 'LineWidth', 2)
plot3(x03(1,:), x03(2,:), x03(3,:), 'b', 'LineWidth', 2)
% Constraint
hc = surf(N,v,Tc, 'EdgeColor', 'none', 'FaceColor', 'r', 'FaceAlpha', 0.5);
% Points
h_mM =
plot3(x_mM(1), x_mM(2), f([x_mM(1), x_mM(2)]), 'o', 'MarkerEdgeColor', 'k', 'MarkerFaceColor', 'c');
h_Mm =
plot3(x_Mm(1), x_Mm(2), f([x_Mm(1), x_Mm(2)]), 'o', 'MarkerEdgeColor', 'k', 'MarkerFaceColor', 'm');
hx =
plot3(x_s(1), x_s(2), f([x_s(1), x_s(2)]), 'o', 'MarkerEdgeColor', 'k', 'MarkerFaceColor', 'r');
% Lines
N1 = zeros(size(N(1,:))); v1 = zeros(size(N(1,:))); l =
zeros(size(N(1,:)));
for i = 1:size(N,2)
    N1(i) = N(1,i); v1(i) = x_mM(2); l(i) = f([N1(i), v1(i)]);
end
plot3(N1, v1, l, 'c--')

N1 = zeros(size(N(1,:))); v1 = zeros(size(N(1,:))); l =
zeros(size(N(1,:)));
for i = 1:size(N,1)
    N1(i) = x_Mm(1); v1(i) = v(i,1); l(i) = f([N1(i), v1(i)]);
end
plot3(N1, v1, l, 'm--')

```

```

N1 = zeros(size(N(1,:))); v1 = zeros(size(N(1,:))); l =
zeros(size(N(1,:)));
for i = 1:size(N,1)
    x = x_s(1) + i*.05*coeff(1,3);
    y = x_s(2) + i*.05*coeff(2,3);
    N1(i) = x; v1(i) = y; l(i) = f([x,y]);
end
plot3(N1,v1,l,'k--')

N1 = zeros(size(N(1,:))); v1 = zeros(size(N(1,:))); l =
zeros(size(N(1,:)));
for i = 1:size(N,1)
    x = x_s(1) - i*.05*coeff(1,3);
    y = x_s(2) - i*.05*coeff(2,3);
    N1(i) = x; v1(i) = y; l(i) = f([x,y]);
end
plot3(N1,v1,l,'k--')

N1 = zeros(size(N(1,:))); v1 = zeros(size(N(1,:))); l =
zeros(size(N(1,:)));
for i = 1:size(N,1)
    x = x_s(1) + i*.05*coeff(1,2);
    y = x_s(2) + i*.05*coeff(2,2);
    N1(i) = x; v1(i) = y; l(i) = f([x,y]);
end
plot3(N1,v1,l,'k--')

N1 = zeros(size(N(1,:))); v1 = zeros(size(N(1,:))); l =
zeros(size(N(1,:)));
for i = 1:size(N,1)
    x = x_s(1) - i*.05*coeff(1,2);
    y = x_s(2) - i*.05*coeff(2,2);
    N1(i) = x; v1(i) = y; l(i) = f([x,y]);
end
plot3(N1,v1,l,'k--')
hold off

xlabel('N','FontSize',fontSize,'FontName','Times New Roman')
ylabel('v (m/s)','FontSize',fontSize,'FontName','Times New Roman')
zlabel('Mapping time (s)','FontSize',fontSize,'FontName','Times New
Roman')
h = legend([hc,h_mM,h_Mm,hx],{'Constraint','Micro-macro
optimum','Macro-micro optimum','Global
optimum'},'location','northeast');
set(h,'FontSize',fontSize,'FontName','Times New Roman')
set(gca,'FontSize',fontSize,'FontName','Times New Roman')
colorbar

% shading interp

set(gca,'ZScale','log')
set(gca,'ZLim',[100 10^4])
% set(gcf,'Position',[1 1 1252 750])
view(115,16)

```



```

%% Plot 3
figure
hold on
costContour = contour(C,[problem.cost.Cmax, problem.cost.Cmax]);
hold off

figure
hold on
contour(C,[40 70 100 110 120])
patch([costContour(1,2:end), 10],[.9+.1*costContour(2,2:end),
11],'red')
hold off

figure
hold on
contourf(N,v,T-1)
% Constraint
plot(costContour(1,2:end),.9+.1*costContour(2,2:end),'r--')
patch([costContour(1,2:end), 10],[.9+.1*costContour(2,2:end),
11],'red','FaceAlpha',0.5)
% Eigenvectors
plot(x01(1,:),x01(2:),'r','LineWidth',2)
plot(x02(1,:),x02(2:),'g','LineWidth',2)
plot(x03(1,:),x03(2:),'b','LineWidth',2)
% Points
h_mM =
plot(x_mM(1),x_mM(2),'o','MarkerEdgeColor','k','MarkerFaceColor','c');
h_Mm =
plot(x_Mm(1),x_Mm(2),'o','MarkerEdgeColor','k','MarkerFaceColor','m');
hx =
plot(x_s(1),x_s(2),'o','MarkerEdgeColor','k','MarkerFaceColor','r');
% Lines
N1 = zeros(size(N(1,:))); v1 = zeros(size(N(1,:))); l =
zeros(size(N(1,:)));
for i = 1:size(N,2)
    N1(i) = N(1,i); v1(i) = x_mM(2); l(i) = f([N1(i),v1(i)]);
end
plot(N1,v1,'c--')

N1 = zeros(size(N(1,:))); v1 = zeros(size(N(1,:))); l =
zeros(size(N(1,:)));
for i = 1:size(N,1)
    N1(i) = x_Mm(1); v1(i) = v(i,1); l(i) = f([N1(i),v1(i)]);
end
plot(N1,v1,'m--')

N1 = zeros(size(N(1,:))); v1 = zeros(size(N(1,:))); l =
zeros(size(N(1,:)));
for i = 1:size(N,1)
    x = x_s(1) + i*.05*coeff(1,3);
    y = x_s(2) + i*.05*coeff(2,3);
    N1(i) = x; v1(i) = y; l(i) = f([x,y]);
end
plot(N1,v1,'k--')

```

```

Nl = zeros(size(N(1,:))); vl = zeros(size(N(1,:))); l =
zeros(size(N(1,:)));
for i = 1:size(N,1)
    x = x_s(1) - i*.05*coeff(1,3);
    y = x_s(2) - i*.05*coeff(2,3);
    Nl(i) = x; vl(i) = y; l(i) = f([x,y]);
end
plot(Nl,vl,'k--')

Nl = zeros(size(N(1,:))); vl = zeros(size(N(1,:))); l =
zeros(size(N(1,:)));
for i = 1:size(N,1)
    x = x_s(1) + i*.05*coeff(1,2);
    y = x_s(2) + i*.05*coeff(2,2);
    Nl(i) = x; vl(i) = y; l(i) = f([x,y]);
end
plot(Nl,vl,'k--')

Nl = zeros(size(N(1,:))); vl = zeros(size(N(1,:))); l =
zeros(size(N(1,:)));
for i = 1:size(N,1)
    x = x_s(1) - i*.05*coeff(1,2);
    y = x_s(2) - i*.05*coeff(2,2);
    Nl(i) = x; vl(i) = y; l(i) = f([x,y]);
end
plot(Nl,vl,'k--')
hold off

xlabel('N','FontSize',fontSize,'FontName','Times New Roman')
ylabel('v (m/s)','FontSize',fontSize,'FontName','Times New Roman')
% h = legend([hc,h_mM,h_Mm,hx],{'Constraint','Micro-macro
optimum','Macro-micro optimum','Global
optimum'},'location','northeast');
% set(h,'FontSize',fontSize,'FontName','Times New Roman')
set(gca,'FontSize',fontSize,'FontName','Times New Roman')
colorbar

% shading interp

%% Analysis
% Rows = {'Macro-micro';'Micro-macro';'Global'};
% fval = [f_Mm;f_mM;f_s];
% cost = [c_Mm;c_mM;c_s];
% N = [x_Mm(1);x_mM(1);x_s(1)];
% v = [x_Mm(2);x_mM(2);x_s(2)];
%
% T = table(N,v,fval,cost,...
%         'RowNames',Rows)

```

micro_Macro_optimizer.m

A sequential micro-macro optimizer for the canonical mapping mission example. Note that some parts are commented to enable the designer to set a different constraint on the system to reflect real-world situations (see section 4.1 page 232).

```
function [x,fval,cost] = micro_Macro_optimizer(problem)
% problem2 = problem;
% problem2.cost.Cmax = 10 + problem.cost.C0;
v = inner_loop(5, problem);
[N,fval,cost] = outer_loop(v, problem);
x = [N,v];
end
```

Macro_micro_optimizer.m

A sequential macro-micro optimizer for the canonical mapping mission example.

```
function [x,fval,cost] = Macro_micro_optimizer(problem)
N = outer_loop(5, problem);
[v,fval,cost] = inner_loop(N, problem);
x = [N,v];
end
```

simultaneous_optimizer.m

A simultaneous optimizer for the canonical mapping mission example.

```
function [x,fval,cost] = simultaneous_optimizer(problem)
% Gather problem data
environment = problem.environment;
cost = problem.cost;

% Prepare GA information
fitnessfcn = @(x)myfitnessfcn(x, environment);
nvars = 2;
A = [];
b = [];
Aeq = [];
beq = [];
LB = [1 1];
UB = [10 10];
nonlcon = @(x)mynonlcon(x, cost);
Intcon = 1;
options = gaoptimset('Generations',1000,...
    'PopulationSize',400,...
    'Display','none');

% Optimize group
```

```

[x, fval] =
ga (fitnessfcn, nvars, A, b, Aeq, beq, LB, UB, nonlcon, Intcon, options);
cost = nonlcon(x) + cost.Cmax;
end

```

outer_loop.m

The outer loop of the simultaneous optimizer.

```

function [N, fval, cost] = outer_loop(v, problem)
% Gather problem data
environment = problem.environment;
cost = problem.cost;

% Prepare GA information
fitnessfcn = @(x)myfitnessfcn([x, v], environment);
nvars = 1;
A = [];
b = [];
Aeq = [];
beq = [];
LB = 1;
UB = 10;
nonlcon = @(x)mynonlcon([x, v], cost);
Intcon = 1;
options = gaoptimset('Generations', 1000, ...
    'PopulationSize', 400, ...
    'Display', 'none');

% Optimize group
[N, fval] =
ga (fitnessfcn, nvars, A, b, Aeq, beq, LB, UB, nonlcon, Intcon, options);
cost = nonlcon(N) + cost.Cmax;
end

```

inner_loop.m

The inner loop of the simultaneous optimizer.

```

function [v, fval, cost] = inner_loop(N, problem)
% Gather problem data
environment = problem.environment;
cost = problem.cost;

% Prepare GA information
fitnessfcn = @(x)myfitnessfcn([N, x], environment);
nvars = 1;
A = [];
b = [];
Aeq = [];

```

```

beq = [];
LB = 1;
UB = 10;
nonlcon = @(x)mynonlcon([N, x], cost);
Intcon = [];
options = gaoptimset('Generations',1000,...
    'PopulationSize',400,...
    'Display','none');

% Optimize group
[v,fval] =
ga(fitnessfcn,nvars,A,b,Aeq,beq,LB,UB,nonlcon,Intcon,options);
cost = nonlcon(v) + cost.Cmax;
end

```

myfitnessfcn.m

The fitness function used by the optimizers to determine the performance of the system.

Here, it is the mapping time.

```

function output = myfitnessfcn(x, environment)
%RUNMISSION Summary of this function goes here
%   Detailed explanation goes here

% INPUT
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Gather input data
d0 = environment.d0;    % Distance from base to mission
lx = environment.lx;    % Map horizontal size
ly = environment.ly;    % Map vertical size

% Parse input vector
N = length(x)/2;
n = x(1:N);             % Number of agents of each type
v = x(N+1:end);         % Velocity for each type of agent

% ANALYSIS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Compute areas for each type of agent -----
---
% ASSUMPTION: each type of agent takes the same amount of time to map
% their given area
S = lx*ly; % Total map area
s = v*S / sum(n.*v);

% Compute mission time for each type of agent
% NOTE: the time to complete the mission is the time for the furthest
agent
% to complete its mission, hence only the furthest agents are studied
here
% Distance to start mission

```

```

ds = d0 + (n-1).*s / ly;

% Distance to map
dm = s;

% Distance to return to base
dr = ds;

% Total distance
d = ds + dm + dr;

% Compute mission time -----
---
output = max(d./v);
end

```

mynonlcon.m

The non-linear constraint function used by the optimizers to constrain the problem. Here, it is the cost of the swarming solution.

```

function [c, ceq] = mynonlcon(x, cost)
% Input
N = x(1);
v = x(2);

% Problem parameters
C0 = cost.C0;           % Group fixed cost
c0 = cost.c0;           % Agents fixed costs
cv = cost.cv;           % Agents linear technology cost
cv2 = cost.cv2;         % Agents quadratic technology cost
Cmax = cost.Cmax;       % Cost constraint

% Compute total group cost
c = C0 + N*(c0 + cv*v + cv2*v^2) - Cmax;
ceq = [];
end

```

B.2.2 Bi-level optimizer

The proposed Matlab implementation of the bi-level genetic algorithm for the optimization of multi-architecture multi-level systems, including the elite retention scheme.

loopOuter.m

The outer loop of the algorithm.

```
function
[xOut, fvalOut, exitflagOut, outputOut, populationOut, scoresOut, ...
    xIn, fvalIn, exitflagIn, outputIn, populationIn, scoresIn] =
loopOuter(optionsOuter, optionsInner)
global buffer elite
if optionsOuter.elitism
    % Initialize elitism retention variables
    buffer = cell(1, optionsOuter.nArchis);
    elite = cell(1, optionsOuter.nArchis);
    optionsInner.elitism = true;
    for i = 1:optionsOuter.nArchis
        buffer{i}.population = [];
        buffer{i}.scores = [];
    end
else
    optionsInner.elitism = false;
end

% Initialization
nArchis = optionsOuter.nArchis;

if strcmp(optionsOuter.solver, 'ga')
    % Define optimization genetic algorithm problem
    problem = struct;
    problem.fitnessfcn = @(x)optionsOuter.fitnessFcn(x, optionsInner);
    problem.nvars = nArchis;
    problem.Aineq = [ones(1, nArchis); -ones(1, nArchis)];
    problem.Bineq = [optionsOuter.maxAgents; -1];
    problem.Aeq = [];
    problem.beq = [];
    problem.lb = zeros(1, nArchis);
    problem.ub = optionsOuter.maxAgents*ones(1, nArchis);
    problem.nonlcon = [];
    problem.intcon = 1:nArchis;
    problem.solver = 'ga';
    problem.options = optionsOuter;

    % Optimize
    [xOut, fvalOut, exitflagOut, outputOut, populationOut, scoresOut] =
ga(problem);
elseif strcmp(optionsOuter.solver, 'ff')
    % Generate full factorial
    dFF = fullfact(optionsOuter.maxAgents*ones(1, nArchis));
    % Keep only groups below maxAgents agents
    indices = sum(dFF, 2) <= optionsOuter.maxAgents;
    dFF = dFF(indices, :);

    % Evaluate fitness over full factorial
    n = size(dFF, 1);
    fitness = zeros(n, 1);
```

```

startTime = tic; % Start timer
if ~strcmp(optionsOuter.Display, 'off')
    fprintf('Started on %s\n', datetime('now'))
end

for i = 1:n
    % Run inner loop
    fitness(i) = optionsOuter.fitnessFcn(dFF(i,:), optionsInner);

    % Estimate remaining time
    elapsed = toc(startTime);
    total = n*elapsed/i;
    remaining = total - elapsed;
    if strcmp(optionsOuter.Display, 'iter')
        fprintf(['%03.0f%% (%d/%d) (elapsed: %fs, remaining: %fs,
total: %fs)\n', 100*i/n, i, n, elapsed, remaining, total)
    end
end

% Display
if ~strcmp(optionsOuter.Display, 'off')
    fprintf('\nFinished on %s\n', datetime('now'));
    fprintf('\tIterations: %d\n', n)
    fprintf('\tFunEval: %d\n', n)
    fprintf('\tTotal time: %fs (%fs/iteration)\n', toc(startTime),
toc(startTime)/n)
end

% Take best architecture
[M,I] = min(fitness);
xOut = dFF(I,:);
fvalOut = M;
exitflagOut = 1;
outputOut = [];
populationOut = dFF;
scoresOut = fitness;
else
    error('Solver argument to outerLoop must be either ''ga'' or
''ff''.')
end

% With known optimal composition, obtain corresponding optimal inner
loop
[fvalIn,xIn,exitflagIn,outputIn,populationIn,scoresIn] =
optionsOuter.fitnessFcn(xOut, optionsInner);
end

```

loopInner.m

The inner loop of the algorithm.


```

function [fval,x,exitflag,output,population,scores] = loopInner(xOuter,
optionsInner)
global elite
% Vector storing architectures in use
archis = 1:length(xOuter);

% Remove absent architectures from the group vectors
toKeep = xOuter > 0;
nVars = optionsInner.nVars;
nVars = nVars(toKeep);
optionsInner.nVars = nVars;
optionsInner.lb = optionsInner.lb(toKeep);
optionsInner.ub = optionsInner.ub(toKeep);
archis = archis(toKeep);
xOuter = xOuter(toKeep);

% Compute number of variables depending on heterogeneity level
if optionsInner.trueHeterogeneity
    nVarsTotalArchis = xOuter.*nVars; % Total variables per
architecture
else
    nVarsTotalArchis = nVars;
end
nVarsTotal = sum(nVarsTotalArchis);

% Compute start indices for each architecture
nArchis = length(xOuter);
startArchis = ones(1,nArchis);
for i = 2:nArchis
    startArchis(i) = startArchis(i-1) + nVarsTotalArchis(i-1);
end

% Compute bounds for the variables
[lb, ub] = createBounds(xOuter, optionsInner.lb, optionsInner.ub,
nVarsTotal, optionsInner.nVars, optionsInner.trueHeterogeneity);

% Compute range for the initial population
optionsInner.PopInitRange = [lb; ub];

% Convey variables to the output function
if optionsInner.elitism == true
    optionsInner.OutputFcns =
{@(options,state,flag)gaInnerLoopElitism(options,state,flag,xOuter,nVar
s,archis,optionsInner.eliteFraction,startArchis,optionsInner.trueHetero
geneity)};
end

% % TODO REMOVE HACK
% optionsInner.OutputFcns =
{@(options,state,flag)gaInnerLoopElitism(options,state,flag,xOuter,nVar
s,archis,optionsInner.eliteFraction,startArchis)};
% % TODO REMOVE HACK - Compute the expected optimum
% offsetsArchis = 1./(1 + (xOuter - archis).^2);
% optionsInner.FitnessLimit = -.99*sum(offsetsArchis);

```

```

% If all elite cells are filled
if optionsInner.elitism == true && sum(cellfun(@isempty,elite)) == 0
    % Initialize population to elite
    optionsInner.InitialPopulation =
generateInitialPopulation(xOuter,elite,nVars,archis,startArchis,nVarsTo
tal,optionsInner.trueHeterogeneity);
end

% Define optimization problem
problem = struct;
problem.fitnessfcn = @(x)optionsInner.fitnessFcn(x, xOuter,
optionsInner.nVars, archis, optionsInner.trueHeterogeneity);
problem.nvars = nVarsTotal;
problem.Aineq = [];
problem.Bineq = [];
problem.Aeq = [];
problem.beq = [];
problem.lb = lb;
problem.ub = ub;
problem.nonlcon = [];
problem.intcon = [];
problem.solver = 'ga';
problem.options = optionsInner;

% Optimize
[x,fval,exitflag,output,population,scores] = ga(problem);
end

function [lb, ub] = createBounds(xOuter, lbArchis, ubArchis,
nVarsTotal, nVars, trueHeterogeneity)
% Initialization
lb = zeros(1, nVarsTotal);
ub = zeros(1, nVarsTotal);

% For each architecture
idx = 1;
for i = 1:length(xOuter)
    % Number of agents for this architecture
    nAgents = xOuter(i);
    % Number of variables for this architecture
    nv = nVars(i);
    % For each variable the architecture
    for k = 1:nv
        if trueHeterogeneity
            % For each agent with this architecture
            for j = 1:nAgents
                % Get upper and lower bounds
                lb(idx) = lbArchis{i}(k);
                ub(idx) = ubArchis{i}(k);

                % Increment array index
                idx = idx + 1;
            end
        else
            % Get upper and lower bounds
            lb(idx) = lbArchis{i}(k);

```

```

        ub(idx) = ubArchis{i}(k);

        % Increment array index
        idx = idx + 1;
    end
end
end
end

function population =
generateInitialPopulation(xOuter,elite,nVars,archis,startArchis,nVarsTotal,trueHeterogeneity)
% Initialization
population = zeros(size(elite{1}.scores,1), nVarsTotal);
nArchis = length(xOuter);

if trueHeterogeneity == true
    % For each architecture
    for i = 1:nArchis
        % For each agent with this architecture
        for j = 1:xOuter(i)
            startIndex = startArchis(i) + (j-1)*nVars(i);
            endIndex = startIndex + nVars(i) - 1;
            population(:,startIndex:endIndex) =
elite{archis(i)}.population;
        end
    end
else
    % For each architecture
    for i = 1:nArchis
        startIndex = startArchis(i);
        endIndex = startIndex + nVars(i) - 1;
        population(:,startIndex:endIndex) =
elite{archis(i)}.population;
    end
end
end
end

```

gaOuterLoopElitism.m

The function runs after each generation of the outer loop. Used for elite retention.

```

function [state,options,optchanged] =
gaOuterLoopElitism(options,state,flag)
% Global variables
global buffer elite
global generationsArray generationOuter
generationOuter = generationOuter + 1;
generationsArray{generationOuter,1} = [];

optchanged = false;
if strcmp(flag,'iter')
    % Update the elite array to buffer

```

```

        elite = buffer;
end
end

```

gaInnerLoopElitism.m

The function runs after each generation of the inner loop. Used for elite retention.

```

function [state,options,optchanged] =
gaInnerLoopElitism(options,state,flag, xOuter, nVars, archis,
eliteFraction, startArchis, trueHeterogeneity)
% Global variables
global buffer
global generationsArray generationOuter
optchanged = false;

if strcmp(flag,'done')
%     xOuter
%     fprintf('[INNER LOOP] Converged in %d
generations\n',state.Generation)

%     nArchis = length(archis);
%     nPopulation = size(state.Population,1);
%     nElite = round(eliteFraction*nPopulation);
%     % For each architecture
%     for i = 1:nArchis
%         if trueHeterogeneity
%             nAgents = xOuter(i); % Number of agents for this
architecture
%         else
%             nAgents = 1;
%         end
%         % Reshape current architecture population
%         temp = zeros(nAgents*nPopulation, nVars(i));
%         for j = 1:nAgents
%             startIndex = startArchis(i) + (j-1)*nVars(i);
%             endIndex = startArchis(i) + j*nVars(i) - 1;
%             temp((1+(j-1)*nPopulation):j*nPopulation,:) =
state.Population(:, startIndex:endIndex);
%         end
%         % Add final population to buffer
%         buffer{archis(i)}.population = [buffer{archis(i)}.population;
temp];
%         % Sort the buffer by corresponding scores
%         scores = [buffer{archis(i)}.scores; repmat(state.Score,
nAgents, 1)];
%         [scores,I] = sort(scores);
%         buffer{archis(i)}.population =
buffer{archis(i)}.population(I,:);
%         % Update current population elite (keep only the required
fraction)
%         buffer{archis(i)}.scores = scores(1:nElite);

```

```

%         buffer{archis(i)}.population =
buffer{archis(i)}.population(1:nElite,:);
%         end

        generationsArray{generationOuter,1} =
[generationsArray{generationOuter,1}, state.Generation];
%         fprintf('-----\n')
end
end

```

gaEstimateTime.m

A function used to estimate the remaining time for the optimization to complete.

```

function [state,options,optchanged] =
gaEstimateTime(options,state,flag)
%GAOUTPUTFCNTEMPLATE Template to write custom OutputFcn for GA.
% [STATE, OPTIONS, OPTCHANGED] =
GAOUTPUTFCNTEMPLATE(OPTIONS,STATE,FLAG)
% where OPTIONS is an options structure used by GA.
%
% STATE: A structure containing the following information about the
state
% of the optimization:
%     Population: Population in the current generation
%     Score: Scores of the current population
%     Generation: Current generation number
%     StartTime: Time when GA started
%     StopFlag: String containing the reason for stopping
%     Selection: Indices of individuals selected for elite,
%               crossover and mutation
%     Expectation: Expectation for selection of individuals
%     Best: Vector containing the best score in each
generation
%     LastImprovement: Generation at which the last improvement in
%                     fitness value occurred
%     LastImprovementTime: Time at which last improvement occurred
%
% FLAG: Current state in which OutputFcn is called. Possible values
are:
%     init: initialization state
%     iter: iteration state
%     interrupt: intermediate state
%     done: final state
%
% STATE: Structure containing information about the state of the
% optimization.
%
% OPTCHANGED: Boolean indicating if the options have changed.
%
% See also PATTERNSEARCH, GA, OPTIMOPTIONS
%
% Copyright 2004-2015 The MathWorks, Inc.

```

```

optchanged = false;

switch flag
    case 'init'
        %       fprintf('Started on %s\n', datestr(datetime('now')))
    case {'iter','interrupt'}
        %       estimated_final_generations = 30;
        %       elapsed = toc(state.StartTime);
        %       n = state.Generation;
        %       total = estimated_final_generations*elapsed/n;
        %       remaining = total - elapsed;
        %       fprintf('Generation %d [%03.0f%%] (elapsed: %fs, remaining:
        %fs, total: %fs, end: %s)\n', n, 100*elapsed/total, elapsed, remaining,
        total, datestr(datetime('now') + seconds(remaining)))
        fprintf('Generation %d\n',state.Generation)
    case 'done'
        %       fprintf('\nFinished on %s\n', datestr(datetime('now')))
        %       fprintf('\tGenerations:      %d\n', state.Generation)
        %       fprintf('\tFunEval:        %d\n', state.FunEval)
        %       fprintf('\tTotal time:      %fs (%fs/generation)\n',
        toc(state.StartTime), toc(state.StartTime)/state.Generation)
        %       fprintf('\tLastImprovement: %d (%fs elapsed since)\n',
        state.LastImprovement, toc(state.LastImprovementTime))
end

```

B.2.3 Unit tests

The unit tests used to validate the optimization algorithm.

test.m

The main file running all the unit tests of the test campaign.

```

% Prepare workspace
clc
close all
clear global
clear

% % Run tests
% result = runtests('outerLoop/outerLoopTest.m');
% rt = table(result);
% disp(rt)
%
% result = runtests('innerLoop/innerLoopTest.m');
% rt = table(result);
% disp(rt)

```

```

result = runtests('completeOptimizer/completeOptimizerTest.m');
rt = table(result);
disp(rt)

```

testFitnessFunctionOuterLoop.m

The fitness function used by the outer loop for the unit tests. The function is described in section 5.2.2.1 page 364.

```

function [fval,x,exitflag,output,population,scores] =
testFitnessFunctionOuterLoop(xOuter, ~)
nArchis = length(xOuter);
% Compute the offset by architecture
offsetsArchis = 1./(1 + (xOuter - (1:nArchis)).^2);
y = - sum(offsetsArchis);

% Assign output variables
fval = y;
x = [];
exitflag = 0;
output = [];
population = [];
scores = [];
end

```

testFitnessFunctionInnerLoop.m

The fitness function used by the inner loop for the unit tests. The function is described in section 5.2.2.1 page 364.

```

function y = testFitnessFunctionInnerLoop(x, xOuter, nVars, archis,
trueHeretogeneity)
% Get number of architectures
nArchis = length(xOuter);

% Prepare offsets
% Each architecture function is a parabola offset by:
% 1) For design variables
%   a) the index of its architecture
%   b) then by a tenth of the number of agents for this architecture
%   b) then by a hundredth of the index of each agent with this
architecture
%   c) then by a thousandth of the index of the design variable for
this agent
% 2) For function value by 1/(1+(nAgents - i)^2) with i index of
architecture

```

```

%
% Example: fitness function for architecture 1, agent 2 is offset by 1
in its values
% and by 1.23 for variable 3, 1.22 for variable 2, etc.
m = sum(xOuter.*nVars);
offsetsVariables = zeros(1,m);
xBis = x;
% For each architecture
idx = 1;
idxArchi = 1;
for i = 1:nArchis
    % Number of agents for this architecture
    nAgents = xOuter(i);
    % Number of variables for this architecture
    nv = nVars(i);
    % For each agent with this architecture
    for j = 1:nAgents
        % For each variable of this agent
        for k = 1:nv
            % Prepare an equivalent design vector
            if trueHeretogeneity
                offsetsVariables(idx) = archis(i) + nAgents/10 + j/100
+ k/1000;
            else
                xBis(idx) = x(idxArchi + k - 1);
                offsetsVariables(idx) = archis(i) + nAgents/10 + 1/100
+ k/1000;
            end
            % Increment array index
            idx = idx + 1;
        end
    end
end

% Increment architecture separation index
if trueHeretogeneity
    % Each architecture has different design variables
    idxArchi = idxArchi + nAgents*nv;
else
    % Similar architectures have the same design variables
    idxArchi = idxArchi + nv;
end
end

% Compute the offset by architecture independently of the number of
variables
offsetsArchis = 1./(1 + (xOuter - archis).^2);

% Compute fitness function
y = sum((xBis - offsetsVariables).^2) - sum(offsetsArchis);
end

```

completeOptimizerTest.m

The unit test for the whole optimizer.


```

%% Main function to generate tests
function tests = completeOptimizerTest
tests = functiontests(localfunctions);
end

%% Test Functions
function testPartialHeterogeneityNoElitism(~)
% Test parameters
nArchis = randi(3);
nVars = randi(5,1,nArchis); % Number of variables per architecture
maxAgents = 6;
% TODO - change between GA and FF
solver = 'ga';

% Perform test
completeOptimizerTestFunction(nArchis, nVars, maxAgents, solver, false,
false, 0)
end

function testTrueHeterogeneityWithElitism(~)
% TODO
end

%% Optional file fixtures
function setupOnce(~) % do not change function name
% Prepare workspace
close all
clear global
clear

addpath('.')
addpath('../..')
end

function teardownOnce(~) % do not change function name
% change back to original path, for example
end

%% Optional fresh fixtures
function setup(~) % do not change function name
end

function teardown(~) % do not change function name
end

```

completeOptimizerTestFunction.m

```

function completeOptimizerTestFunction(nArchis, nVars, maxAgents,
solver, ...
    trueHeterogeneity, elitism, elitismFraction)
% Outer loop
=====

```

```

optionsOuter = gaoptimset(...
    'Display',          'off',...
    'OutputFcns',      {@gaEstimateTime},...
    'PlotFcn',         {[]},...
    'PlotInterval',   1,...
    'PopulationSize',  50,...
    'TolFun',          1e-6,...
    'UseParallel',     false);

optionsOuter.nArchis = nArchis; % Number of architectures
optionsOuter.maxAgents = maxAgents; % Maximum number of agents in the
group
optionsOuter.solver = solver;
optionsOuter.elitism = elitism;
optionsOuter.elitismFraction = elitismFraction;
optionsOuter.fitnessFcn = @loopInner;

% Inner loop
=====
optionsInner = gaoptimset(...
    'TolCon',          1e-3,...
    'Display',         'none',...
    'OutputFcns',      {[]},...
    'PlotFcn',         {[]},...
    'PlotInterval',   1,...
    'PopulationSize',  200,...
    'TolFun',          1e-6,...
    'UseParallel',     false);

% Custom options
optionsInner.fitnessFcn = @testFitnessFunctionInnerLoop;
optionsInner.trueHeterogeneity = trueHeterogeneity; % Regroup per
architecture or not
optionsInner.nVars = nVars; % Number of variables per architecture
optionsInner.elitism = false;

% Bounds
for i = 1:length(nVars)
    % Lower bounds
    optionsInner.lb{i} = zeros(1, nVars(i));
    % Upper bounds
    optionsInner.ub{i} = (nArchis + 3)*ones(1, nVars(i));
end

% Optimize
=====
[xOut,~,~,~,~,~,xIn] = loopOuter(optionsOuter, optionsInner);

% Test outer loop
=====
assert(sum(xOut - (1:nArchis)) == 0)

% Test inner loop
=====
xOuter = xOut;

```

```

% Keep only non 0 architectures
nVars = nVars(xOuter > 0);
xOuter = xOuter(xOuter > 0);
tolerance = 1e-2;
nArchis = length(xOuter);

if trueHeterogeneity
    m = sum(xOuter.*nVars);
else
    m = sum(nVars);
end
xExpected = zeros(1,m);
% For each architecture
idx = 1;
for i = 1:nArchis
    % Number of agents for this architecture
    nAgents = xOuter(i);
    % Number of variables for this architecture
    nv = nVars(i);
    % For each agent with this architecture
    if trueHeterogeneity
        for j = 1:nAgents
            % For each variable of this agent
            for k = 1:nv
                % Prepare an analytical optimum vector
                xExpected(idx) = i + nAgents/10 + j/100 + k/1000;
                % Increment array index
                idx = idx + 1;
            end
        end
    else
        for k = 1:nv
            % Prepare an analytical optimum vector
            xExpected(idx) = i + nAgents/10 + 1/100 + k/1000;
            % Increment array index
            idx = idx + 1;
        end
    end
end
end

% Compare expected and actual optimum
assert(mean((xIn - xExpected).^2) < tolerance, sprintf('Norm2 error is
%f', mean((xIn - xExpected).^2)))
end

```

outerLoopTest.m

The unit test for the outer loop.

```

%% Main function to generate tests
function tests = outerLoopTest
tests = functiontests(localfunctions);

```

```

end

%% Test Functions
function testFFPartialHeterogeneity(~)
% Test parameters
nArchis = randi(3);
maxAgents = 6;
solver = 'ff';

% Perform test
outerLoopTestFunction(nArchis, maxAgents, solver)
end

function testGAPartialHeterogeneity(~)
% Test parameters
nArchis = randi(10);
maxAgents = sum(1:nArchis);
solver = 'ga';

% Perform test
outerLoopTestFunction(nArchis, maxAgents, solver)
end

%% Optional file fixtures
function setUpOnce(~) % do not change function name
% Prepare workspace
close all
clear global
clear

addpath('.')
addpath('../..')
end

function tearDownOnce(~) % do not change function name
% change back to original path, for example
end

%% Optional fresh fixtures
function setUp(~) % do not change function name
end

function tearDown(~) % do not change function name
end

```

outerLoopTestFunction.m

```

function outerLoopTestFunction(nArchis, maxAgents, solver)
% Outer loop
=====
optionsOuter = gaoptimset(...
    'Display', 'off', ...

```

```

    'OutputFcns',      {},...
    'PlotFcn',        {},...
    'PlotInterval',   1,...
    'PopulationSize', 50,...
    'TolFun',         1e-6,...
    'UseParallel',    false);

optionsOuter.nArchis = nArchis; % Number of architectures
optionsOuter.maxAgents = maxAgents; % Maximum number of agents in the
group
optionsOuter.solver = solver;
optionsOuter.elitism = false;
optionsOuter.fitnessFcn = @testFitnessFunctionOuterLoop;

% Inner loop
=====
optionsInner = [];

% Optimize
=====
[x,~,~,~,~,~] = loopOuter(optionsOuter, optionsInner);

% Test
assert(sum(x - (1:nArchis)) == 0)
end

```

innerLoopTest.m

The unit test for the inner loop.

```

%% Main function to generate tests
function tests = innerLoopTest
tests = functiontests(localfunctions);
end

%% Test Functions
function testPartialHeterogeneity(~)
innerLoopTestFunction(false)
end

function testFullHeterogeneity(~)
innerLoopTestFunction(true)
end

%% Optional file fixtures
function setupOnce(~) % do not change function name
% Prepare workspace
close all
clear global
clear

addpath('..')

```

```

addpath('../..')
end

function teardownOnce(~) % do not change function name
% change back to original path, for example
end

%% Optional fresh fixtures
function setup(~) % do not change function name
end

function teardown(~) % do not change function name
end

```

innerLoopTestFunction.m

```

function innerLoopTestFunction(trueHeterogeneity)
% General scope variables
=====
% Generate random group with random architectures
nArchis = randi(5);
xOuter = randi([0 5],1,nArchis); % Number of agents for each
architecture
nVars = randi(5,1,nArchis); % Number of variables per architecture

% Ensure that there is at least one agent in the group
while sum(xOuter) < 1
    nArchis = randi(5);
    xOuter = randi([0 5],1,nArchis); % Number of agents for each
architecture
    nVars = randi(5,1,nArchis); % Number of variables per architecture
end

% Inner loop options
=====
optionsInner = gaoptimset(...
    'TolCon',          1e-3,...
    'Display',         'none',...
    'OutputFcns',      {},...
    'PlotFcn',         {},...
    'PlotInterval',   1,...
    'PopulationSize',  200,...
    'TolFun',          1e-6,...
    'UseParallel',    false);

% Custom options
optionsInner.fitnessFcn = @testFitnessFunctionInnerLoop;
optionsInner.trueHeterogeneity = trueHeterogeneity; % Regroup per
architecture or not
optionsInner.nVars = nVars; % Number of variables per architecture
optionsInner.elitism = false;

% Bounds

```

```

for i = 1:length(nVars)
    % Lower bounds
    optionsInner.lb{i} = zeros(1, nVars(i));
    % Upper bounds
    optionsInner.ub{i} = (nArchis + 3)*ones(1, nVars(i));
end

% Optimize
=====
[~,x,~,~,~,~] = loopInner(xOuter, optionsInner);

% Test
=====

% Keep only non 0 architectures
nVars = nVars(xOuter > 0);
xOuter = xOuter(xOuter > 0);
tolerance = 1e-2;
nArchis = length(xOuter);

if trueHeterogeneity
    m = sum(xOuter.*nVars);
else
    m = sum(nVars);
end
xExpected = zeros(1,m);
% For each architecture
idx = 1;
for i = 1:nArchis
    % Number of agents for this architecture
    nAgents = xOuter(i);
    % Number of variables for this architecture
    nv = nVars(i);
    % For each agent with this architecture
    if trueHeterogeneity
        for j = 1:nAgents
            % For each variable of this agent
            for k = 1:nv
                % Prepare an analytical optimum vector
                xExpected(idx) = i + nAgents/10 + j/100 + k/1000;
                % Increment array index
                idx = idx + 1;
            end
        end
    else
        for k = 1:nv
            % Prepare an analytical optimum vector
            xExpected(idx) = i + nAgents/10 + 1/100 + k/1000;
            % Increment array index
            idx = idx + 1;
        end
    end
end
end

% Compare expected and actual optimum
assert(mean((x - xExpected).^2) < tolerance, sprintf('Norm2 error is
%f', mean((x - xExpected).^2)))

```

```
end
```

B.2.4 Plots for the test function

The scripts used to analyze the behavior of the test function described in section 5.2.2.1 page 364.

plotOuterLoopOffsets.m

A plot showing how the function is affected by changes in the outer loop design vector.

```
% Prepare workspace
clc
close all
clear global
clear

offset = zeros(5,3);
for i = 1:3
    for j = 1:5
        % Compute offset
        Xout = [i j];
        archis = 1:length(Xout);
        offset(j,i) = -sum(1./(1 + (Xout - archis).^2));
    end
end

N1 = 1:3;
N2 = 1:0.1:5;
offset_continuous = zeros(length(N2),length(N1));
for i = 1:length(N1)
    for j = 1:length(N2)
        % Compute offset
        Xout = [N1(i) N2(j)];
        archis = 1:length(Xout);
        offset_continuous(j,i) = -sum(1./(1 + (Xout - archis).^2));
    end
end

%% Plot
figure
c = get(gca, 'colororder');

hold on

plot(N2,offset_continuous,'--')
```



```

b = bar(offset);
b(1).FaceColor = c(1,:);
b(2).FaceColor = c(2,:);
b(3).FaceColor = c(3,:);

hold off
xlabel('N_2','FontName','Times New Roman','FontSize',12)
ylabel('f_{out}(X_{out})','FontName','Times New Roman','FontSize',12)
h = legend(b,{'N_1 = 1','N_1 = 2','N_1 = 3'},'Location','SouthEast');
set(h,'FontName','Times New Roman','FontSize',12)
set(gca,'FontName','Times New Roman','FontSize',12)

```

plotOuterLoop.m

A plot of the outer loop part of the test function.

```

% Prepare workspace
clc
close all
clear global
clear

addpath('../models')
addpath('../optimizer')

% Number of architectures
nArchis = 2;
archis = [1 2];

% Out-of-loop variables
=====
nVars = [1 1]; % Number of variables per architecture

% Inner loop options
optionsInner = gaoptimset(...
    'TolCon',          1e-3,...
    'Display',         'none',...
    'OutputFcns',     {[]},...
    'PlotFcn',         {[]},...
    'PlotInterval',   1,...
    'PopulationSize', 200,...
    'TolFun',          1e-6,...
    'UseParallel',    false);

% Custom options
optionsInner.fitnessFcn = @testFitnessFunctionInnerLoop;
optionsInner.trueHeterogeneity = false; % Regroup per architecture or
not
optionsInner.nVars = nVars; % Number of variables per architecture
optionsInner.elitism = false;

% Bounds

```

```

for i = 1:length(nVars)
    % Lower bounds
    optionsInner.lb{i} = zeros(1, nVars(i));
    % Upper bounds
    optionsInner.ub{i} = (nArchis + 3)*ones(1, nVars(i));
end

n1Max = 5;
n2Max = 5;
[N1,N2] = meshgrid(1:n1Max, 1:n2Max);
Xopt = cell(n1Max, n2Max);
Fval = zeros(n1Max, n2Max);
for m = 1:size(N1,1)
    for n = 1:size(N1,2)
        fprintf('Case %d/%d\n', n+(m-1)*n1Max, n1Max*n2Max)

        xOuter = [N1(m,n) N2(m,n)]; % Number of agents for each
architecture

        % Compute range for the initial population -----
----
        % Remove absent architectures from the group vectors
        nVarsBis = nVars(xOuter > 0);
        xOuterBis = xOuter(xOuter > 0);

        % Compute number of variables depending on heterogeneity level
        if optionsInner.trueHeterogeneity
            nVarsTotal = sum(xOuterBis.*nVarsBis);
        else
            nVarsTotal = sum(nVarsBis);
        end

        % Compute bounds for the variables
        lb = zeros(1, nVarsTotal);
        ub = zeros(1, nVarsTotal);
        % For each architecture
        idx = 1;
        for i = 1:length(xOuter)
            % Number of agents for this architecture
            nAgents = xOuter(i);
            % Number of variables for this architecture
            nv = nVars(i);
            % For each variable the architecture
            for k = 1:nv
                if optionsInner.trueHeterogeneity
                    % For each agent with this architecture
                    for j = 1:nAgents
                        % Get upper and lower bounds
                        lb(idx) = optionsInner.lb{i}(k);
                        ub(idx) = optionsInner.ub{i}(k);

                        % Increment array index
                        idx = idx + 1;
                    end
                else
                    % Get upper and lower bounds

```

```

        lb(idx) = optionsInner.lb{i}(k);
        ub(idx) = optionsInner.ub{i}(k);

        % Increment array index
        idx = idx + 1;
    end
end
end

optionsInner.PopInitRange = [lb; ub];

% Optimize -----
---
[fval,x_opt,~,~,~,~] = loopInner2(xOuter, optionsInner);

% Store results -----
---
Xopt{m,n} = x_opt;
Fval(m,n) = fval;
end
end

%% Plot
=====

res = 0.01;

% Architecture 1 -----
---
figure
hold on
my_legend = cell(1,nlMax+1);
my_legend{end} = 'Optima';
p = zeros(1, nlMax+1);

m = 1; % Number of architecture 2
for n = 1:nlMax
    xOpt = Xopt{m,n};
    fval = Fval(m,n);
    x1 = lb(1):res:ub(1);
    x2 = xOpt(2);
    y = zeros(size(x1));
    xOuter = [N1(m,n), N2(m,n)];
    for i = 1:length(x1)
        y(i) = optionsInner.fitnessFcn([x1(i), x2], xOuter, nVars,
archis, optionsInner.trueHeterogeneity);
    end

    p(end) = plot(xOpt(1),fval,'*r');
    plot([xOpt(1), xOpt(1)], [10, fval], '--', 'color', 0.8*ones(3,1))
    plot([0, xOpt(1)], [fval, fval], '--', 'color', 0.8*ones(3,1))
    p(n) = plot(x1,y);
    text(xOpt(1), 10.3, sprintf('%3.3f',
xOpt(1)), 'Color', 0.3*ones(3,1), 'HorizontalAlignment', 'left', 'Rotation',
45)

```

```

    text(0, fval, sprintf('%3.1f', fval),
'Color',0.3*ones(3,1), 'HorizontalAlignment', 'left')
    my_legend{n} = sprintf('n_1 = %d', n);
end

hold off
xlabel('x_1')
ylabel('y')
legend(p, my_legend, 'Location', 'northeast')
axis([0.25 2.5 -10 10])

% % Architecture 2 -----
-----
figure
hold on
my_legend = cell(1,n2Max+1);
my_legend{end} = 'Optima';
p = zeros(1, n2Max+1);

n = 1; % Number of architecture 2
for m = 1:n2Max
    xOpt = Xopt{m,n};
    fval = Fval(m,n);
    x1 = xOpt(1);
    x2 = lb(1):res:ub(1);
    y = zeros(size(x2));
    xOuter = [N1(m,n), N2(m,n)];
    for i = 1:length(x2)
        y(i) = optionsInner.fitnessFcn([x1, x2(i)], xOuter, nVars,
archis, optionsInner.trueHeterogeneity);
    end

    p(end) = plot(xOpt(2), fval, '*r');
    plot([xOpt(2), xOpt(2)], [20, fval], '--', 'color', 0.8*ones(3,1))
    plot([0, xOpt(2)], [fval, fval], '--', 'color', 0.8*ones(3,1))
    p(m) = plot(x2, y);
    text(xOpt(2), 21, sprintf('%3.3f',
xOpt(2)), 'Color', 0.3*ones(3,1), 'HorizontalAlignment', 'left', 'Rotation',
45)
    text(0, fval, sprintf('%3.1f', fval),
'Color', 0.3*ones(3,1), 'HorizontalAlignment', 'left')
    my_legend{m} = sprintf('n_2 = %d', m);
end

hold off
xlabel('x_2')
ylabel('y')
legend(p, my_legend, 'Location', 'southeast')
% axis([0.5 5 -15 20])

```

plotInnerLoop.m

A display of how the test function values are affected by the inner loop design vector.

```

% Prepare workspace
clc
close all
clear global
clear

addpath('.././models')
addpath('.././optimizer')

% General scope variables
=====
% Generate random group with random architectures
nArchis = 2;
xOuter = [3 2]; % Number of agents for each architecture
archis = [1 2]; % Architecture indices
nVars = [1 1]; % Number of variables per architecture

% Inner loop options
=====
optionsInner = gaoptimset(...
    'TolCon',          1e-3,...
    'Display',         'none',...
    'OutputFcns',     {[]},...
    'PlotFcn',         {[]},...
    'PlotInterval',   1,...
    'PopulationSize', 200,...
    'TolFun',          1e-6,...
    'UseParallel',    false);

% Custom options
optionsInner.fitnessFcn = @testFitnessFunctionInnerLoop;
optionsInner.trueHeterogeneity = false; % Regroup per architecture or
not
optionsInner.nVars = nVars; % Number of variables per architecture
optionsInner.elitism = false;

% Bounds
for i = 1:length(nVars)
    % Lower bounds
    optionsInner.lb{i} = zeros(1, nVars(i));
    % Upper bounds
    optionsInner.ub{i} = (nArchis + 3)*ones(1, nVars(i));
end

% Compute range for the initial population -----
---
% Remove absent architectures from the group vectors
nVarsBis = nVars(xOuter > 0);
xOuterBis = xOuter(xOuter > 0);

% Compute number of variables depending on heterogeneity level
if optionsInner.trueHeterogeneity
    nVarsTotal = sum(xOuterBis.*nVarsBis);
else
    nVarsTotal = sum(nVarsBis);

```

```

end

% Compute bounds for the variables
lb = zeros(1, nVarsTotal);
ub = zeros(1, nVarsTotal);
% For each architecture
idx = 1;
for i = 1:length(xOuter)
    % Number of agents for this architecture
    nAgents = xOuter(i);
    % Number of variables for this architecture
    nv = nVars(i);
    % For each variable the architecture
    for k = 1:nv
        if optionsInner.trueHeterogeneity
            % For each agent with this architecture
            for j = 1:nAgents
                % Get upper and lower bounds
                lb(idx) = optionsInner.lb{i}(k);
                ub(idx) = optionsInner.ub{i}(k);

                % Increment array index
                idx = idx + 1;
            end
        else
            % Get upper and lower bounds
            lb(idx) = optionsInner.lb{i}(k);
            ub(idx) = optionsInner.ub{i}(k);

            % Increment array index
            idx = idx + 1;
        end
    end
end
end

optionsInner.PopInitRange = [lb; ub];

% Optimize
=====
[fval,xOpt,~,~,~,~] = loopInner2(xOuter, optionsInner);

%% Plot
=====

res = 0.01;

% Architecture 1 -----
----
x1 = lb(1):res:ub(1);
x2 = xOpt(2);
y = zeros(size(x1));
for i = 1:length(x1)
    y(i) = optionsInner.fitnessFcn([x1(i), x2], xOuter, nVars, archis,
optionsInner.trueHeterogeneity);
end

```

```

figure
hold on
plot(xOpt(1), fval, '*r')
plot(x1, y)
plot([xOpt(1), xOpt(1)], [-10, fval], '--r')
plot([0, xOpt(1)], [fval, fval], '--r')
text(xOpt(1), -14, sprintf('%3.3f', xOpt(1)), 'Color', 'r',
'HorizontalAlignment', 'center', 'FontName', 'Times New
Roman', 'FontSize', 12)
text(-0.3, fval - 0.5, sprintf('%3.2f', fval), 'Color', 'r',
'HorizontalAlignment', 'center', 'FontName', 'Times New
Roman', 'FontSize', 12)
hold off
xlabel('x_1', 'FontName', 'Times New Roman', 'FontSize', 12)
ylabel('y', 'FontName', 'Times New Roman', 'FontSize', 12)
h = legend('Optimum');
set(h, 'FontName', 'Times New Roman', 'FontSize', 12)
set(gca, 'FontName', 'Times New Roman', 'FontSize', 12)

% Architecture 2 -----
---
x1 = xOpt(1);
x2 = lb(1):res:ub(1);
y = zeros(size(x2));
for i = 1:length(x2)
    y(i) = optionsInner.fitnessFcn([x1, x2(i)], xOuter, nVars, archis,
optionsInner.trueHeterogeneity);
end

figure
hold on
plot(xOpt(2), fval, '*r')
plot(x2, y)
plot([xOpt(2), xOpt(2)], [-10, fval], '--r')
plot([0, xOpt(2)], [fval, fval], '--r')
text(xOpt(2), -12.0, sprintf('%3.3f', xOpt(2)), 'Color', 'r',
'HorizontalAlignment', 'center', 'FontName', 'Times New
Roman', 'FontSize', 12)
text(-0.3, fval, sprintf('%3.2f', fval), 'Color', 'r',
'HorizontalAlignment', 'center', 'FontName', 'Times New
Roman', 'FontSize', 12)
hold off
xlabel('x_2', 'FontName', 'Times New Roman', 'FontSize', 12)
ylabel('y', 'FontName', 'Times New Roman', 'FontSize', 12)
h = legend('Optimum');
set(h, 'FontName', 'Times New Roman', 'FontSize', 12)
set(gca, 'FontName', 'Times New Roman', 'FontSize', 12)

% Fitness function -----
---
[X1, X2] = meshgrid(lb(1):res:ub(1), lb(1):res:ub(2));
Y = zeros(size(X1));
for i = 1:size(X1,1)
    for j = 1:size(X1,2)
        Y(i,j) = optionsInner.fitnessFcn([X1(i,j), X2(i,j)], xOuter,
nVars, archis, optionsInner.trueHeterogeneity);
    end
end

```

```

    end
end

figure
hold on
[C,h] = contourf(X1,X2,Y,10);
clabel(C,h,'FontName','Times New Roman','FontSize',12)
plot(xOpt(1), xOpt(2),'*r')
plot([xOpt(1), xOpt(1)], [0, xOpt(2)], '--r')
plot([0, xOpt(1)], [xOpt(2), xOpt(2)], '--r')
text(-0.4, xOpt(2), sprintf('%3.3f', xOpt(2)), 'Color', 'r',
'HorizontalAlignment', 'center','FontName','Times New
Roman','FontSize',12)
text(xOpt(1), -0.4, sprintf('%3.3f', xOpt(1)), 'Color', 'r',
'HorizontalAlignment', 'center','FontName','Times New
Roman','FontSize',12)
text(1.05*xOpt(1), 1.05*xOpt(2), sprintf('%3.2f', fval), 'Color', 'k',
'HorizontalAlignment', 'left','FontName','Times New
Roman','FontSize',12)
hold off
xlabel('x_1','FontName','Times New Roman','FontSize',12)
ylabel('x_2','FontName','Times New Roman','FontSize',12)
h = legend('Test function contours', 'Optimum');
set(h,'FontName','Times New Roman','FontSize',12)
set(gca,'FontName','Times New Roman','FontSize',12)
axis equal

print -dmeta -r600;

```

plotConstraintInnerLoop.m

A display of how the optimum of the test function is affected when the problem is constrained. The plot is done only for the inner loop part of the test function.

```

% Prepare workspace
clc
close all
clear global
clear

addpath('../../models')
addpath('../../optimizer')

% General scope variables
=====
% Generate random group with random architectures
nArchis = 2;
xOuter = [3 2]; % Number of agents for each architecture
archis = [1 2]; % Architecture indices
nVars = [1 1]; % Number of variables per architecture

```



```

% Inner loop options
=====
optionsInner = gaoptimset(...
    'TolCon',          1e-3,...
    'Display',         'none',...
    'OutputFcns',     {[]},...
    'PlotFcn',         {[]},...
    'PlotInterval',   1,...
    'PopulationSize', 200,...
    'TolFun',          1e-6,...
    'UseParallel',    false);

% Custom options
optionsInner.fitnessFcn = @testFitnessFunctionInnerLoop;
optionsInner.trueHeterogeneity = false; % Regroup per architecture or
not
optionsInner.nVars = nVars; % Number of variables per architecture
optionsInner.elitism = false;

% Bounds
for i = 1:length(nVars)
    % Lower bounds
    optionsInner.lb{i} = zeros(1, nVars(i));
    % Upper bounds
    optionsInner.ub{i} = (nArchis + 3)*ones(1, nVars(i));
end

% Compute range for the initial population -----
---
% Remove absent architectures from the group vectors
nVarsBis = nVars(xOuter > 0);
xOuterBis = xOuter(xOuter > 0);

% Compute number of variables depending on heterogeneity level
if optionsInner.trueHeterogeneity
    nVarsTotal = sum(xOuterBis.*nVarsBis);
else
    nVarsTotal = sum(nVarsBis);
end

% Compute bounds for the variables
lb = zeros(1, nVarsTotal);
ub = zeros(1, nVarsTotal);
% For each architecture
idx = 1;
for i = 1:length(xOuter)
    % Number of agents for this architecture
    nAgents = xOuter(i);
    % Number of variables for this architecture
    nv = nVars(i);
    % For each variable the architecture
    for k = 1:nv
        if optionsInner.trueHeterogeneity
            % For each agent with this architecture
            for j = 1:nAgents
                % Get upper and lower bounds

```

```

        lb(idx) = optionsInner.lb{i}(k);
        ub(idx) = optionsInner.ub{i}(k);

        % Increment array index
        idx = idx + 1;
    end
else
    % Get upper and lower bounds
    lb(idx) = optionsInner.lb{i}(k);
    ub(idx) = optionsInner.ub{i}(k);

    % Increment array index
    idx = idx + 1;
end
end
end

optionsInner.PopInitRange = [lb; ub];

% Optimize
=====
[fval,xOpt,~,~,~,~] = loopInner2(xOuter, optionsInner);

%% Plot
=====

res = 0.01;

% Architecture 1 -----
---
x1 = lb(1)-1:res:ub(1)+1;
x2 = xOpt(2)+1;
y = zeros(size(x1));
for i = 1:length(x1)
    y(i) = optionsInner.fitnessFcn([x1(i), x2], xOuter, nVars, archis,
optionsInner.trueHeterogeneity);
end

fval = optionsInner.fitnessFcn([xOpt(1), xOpt(2)+1], xOuter, nVars,
archis, optionsInner.trueHeterogeneity);
fval2 = optionsInner.fitnessFcn([xOpt(1)+1, x2], xOuter, nVars, archis,
optionsInner.trueHeterogeneity);

figure
hold on
plot(xOpt(1), fval, '*r')
plot(xOpt(1) + 1, fval2, '*m')
plot(x1,y)
plot([xOpt(1), xOpt(1)],[-10, fval], '--r')
plot([lb(1)-1, xOpt(1)], [fval, fval], '--r')
text(xOpt(1), -17, sprintf('%3.3f', xOpt(1)), 'Color', 'r',
'HorizontalAlignment', 'center', 'FontName', 'Times New
Roman', 'FontSize', 12)
text(lb(1)-1-0.5, fval - 0.5, sprintf('%3.2f', fval), 'Color', 'r',
'HorizontalAlignment', 'center', 'FontName', 'Times New
Roman', 'FontSize', 12)

```

```

plot([xOpt(1) + 1, xOpt(1) + 1],[-10, 70],'-m')
plot([lb(1)-1, xOpt(1)+1],[fval2, fval2], '--m')
text(xOpt(1) + 1, -17, sprintf('%3.3f', xOpt(1) + 1), 'Color', 'm',
'HorizontalAlignment', 'center','FontName','Times New
Roman','FontSize',12)
text(lb(1)-1-0.5, fval2 - 0.5, sprintf('%3.2f', fval2), 'Color', 'm',
'HorizontalAlignment', 'center','FontName','Times New
Roman','FontSize',12)
hold off
xlabel('x_1','FontName','Times New Roman','FontSize',12)
ylabel('y','FontName','Times New Roman','FontSize',12)
h = legend('Unconstrained optimum','Constrained
optimum','location','northoutside');
set(h,'FontName','Times New Roman','FontSize',12)
set(gca,'FontName','Times New Roman','FontSize',12)

% Architecture 2 -----
---
x1 = xOpt(1)+1;
x2 = lb(1)-1:res:ub(1)+1;
y = zeros(size(x2));
for i = 1:length(x2)
    y(i) = optionsInner.fitnessFcn([x1, x2(i)], xOuter, nVars, archis,
optionsInner.trueHeterogeneity);
end

fval = optionsInner.fitnessFcn([xOpt(1)+1, xOpt(2)], xOuter, nVars,
archis, optionsInner.trueHeterogeneity);
fval3 = optionsInner.fitnessFcn([x1, xOpt(2)+1], xOuter, nVars, archis,
optionsInner.trueHeterogeneity);

figure
hold on
plot(xOpt(2), fval, '*r')
plot(xOpt(2)+1, fval3, '*m')
plot(x2, y)
plot([xOpt(2), xOpt(2)],[-10, fval], '--r')
plot([lb(2)-1, xOpt(2)], [fval3, fval3], '--r')
text(xOpt(2), -15, sprintf('%3.3f', xOpt(2)), 'Color', 'r',
'HorizontalAlignment', 'center','FontName','Times New
Roman','FontSize',12)
text(lb(2)-1-0.5, fval, sprintf('%3.2f', fval), 'Color', 'r',
'HorizontalAlignment', 'center','FontName','Times New
Roman','FontSize',12)

plot([xOpt(2) + 1, xOpt(2) + 1],[-10, 35],'-m')
plot([lb(2)-1, xOpt(2)+1],[fval3, fval3], '--m')
text(xOpt(2) + 1, -15, sprintf('%3.3f', xOpt(2) + 1), 'Color', 'm',
'HorizontalAlignment', 'center','FontName','Times New
Roman','FontSize',12)
text(lb(2)-1-0.5, fval3, sprintf('%3.2f', fval3), 'Color', 'm',
'HorizontalAlignment', 'center','FontName','Times New
Roman','FontSize',12)
hold off
xlabel('x_2','FontName','Times New Roman','FontSize',12)

```

```

ylabel('y','FontName','Times New Roman','FontSize',12)
h = legend('Unconstrained optimum','Constrained
optimum','location','northoutside');
set(h,'FontName','Times New Roman','FontSize',12)
set(gca,'FontName','Times New Roman','FontSize',12)

% Fitness function -----
---
[X1, X2] = meshgrid(lb(1)-1:res:ub(1)+1, lb(1)-1:res:ub(2)+1);
Y = zeros(size(X1));
for i = 1:size(X1,1)
    for j = 1:size(X1,2)
        Y(i,j) = optionsInner.fitnessFcn([X1(i,j), X2(i,j)], xOuter,
nVars, archis, optionsInner.trueHeterogeneity);
    end
end

figure
hold on
[C,h] = contourf(X1,X2,Y,10);
clabel(C,h,'FontName','Times New Roman','FontSize',12)
plot(xOpt(1), xOpt(2),'*r')
plot(xOpt(1)+1, xOpt(2)+1,'*m')
plot([xOpt(1), xOpt(1)], [lb(2)-1, xOpt(2)], '--r')
plot([lb(1)-1, xOpt(1)], [xOpt(2), xOpt(2)], '--r')
text(lb(1)-1-0.4, xOpt(2), sprintf('%3.3f', xOpt(2)), 'Color', 'r',
'HorizontalAlignment', 'center','FontName','Times New
Roman','FontSize',12)
text(xOpt(1), lb(2)-1-.55, sprintf('%3.3f', xOpt(1)), 'Color', 'r',
'HorizontalAlignment', 'center','FontName','Times New
Roman','FontSize',12)
text(1.05*xOpt(1), 1.05*xOpt(2), sprintf('%3.2f', fval), 'Color', 'r',
'HorizontalAlignment', 'left','FontName','Times New
Roman','FontSize',12)

plot([xOpt(1)+1, xOpt(1)+1], [lb(2)-1, xOpt(2)+1], '-m')
plot([lb(1)-1, xOpt(1)+1], [xOpt(2)+1, xOpt(2)+1], '-m')
text(lb(1)-1-0.4, xOpt(2)+1, sprintf('%3.3f', xOpt(2)+1), 'Color', 'm',
'HorizontalAlignment', 'center','FontName','Times New
Roman','FontSize',12)
text(xOpt(1)+1, lb(2)-1-0.55, sprintf('%3.3f', xOpt(1)+1), 'Color',
'm', 'HorizontalAlignment', 'center','FontName','Times New
Roman','FontSize',12)
text(1.05*xOpt(1)+1, 1.05*xOpt(2)+1, sprintf('%3.2f', fval3), 'Color',
'm', 'HorizontalAlignment', 'left','FontName','Times New
Roman','FontSize',12)
hold off
xlabel('x_1','FontName','Times New Roman','FontSize',12)
ylabel('x_2','FontName','Times New Roman','FontSize',12)
h = legend('Test function contours', 'Unconstrained optimum',
'Constrained optimum','location','northoutside');
set(h,'FontName','Times New Roman','FontSize',12)
set(gca,'FontName','Times New Roman','FontSize',12)
axis equal

print -dmeta -r600;

```

loopOuter2.m

A modified version of the outer loop of the optimization algorithm to be used in the developing phase.

```
function [x,fval,exitflag,output,population,scores] =
loopOuter2(optionsOuter, optionsInner)
global buffer elite
if optionsOuter.elitism
    % Initialize elitism retention variables
    buffer = cell(1, optionsOuter.nArchis);
    elite = cell(1, optionsOuter.nArchis);
    optionsInner.elitism = true;
    for i = 1:optionsOuter.nArchis
        buffer{i}.population = [];
        buffer{i}.scores = [];
    end
else
    optionsInner.elitism = false;
end

% Initialization
nArchis = optionsOuter.nArchis;

if strcmp(optionsOuter.solver, 'ga')
    % Define optimization genetic algorithm problem
    problem = struct;
    problem.fitnessfcn = @(x)optionsOuter.fitnessFcn(x, optionsInner);
    problem.nvars = nArchis;
    problem.Aineq = [ones(1, nArchis);-ones(1,nArchis)];
    problem.Bineq = [optionsOuter.maxAgents; -1];
    problem.Aeq = [];
    problem.beq = [];
    problem.lb = zeros(1, nArchis);
    problem.ub = optionsOuter.maxAgents*ones(1, nArchis);
    problem.nonlcon = [];
    problem.intcon = 1:nArchis;
    problem.solver = 'ga';
    problem.options = optionsOuter;

    % Optimize
    [x,fval,exitflag,output,population,scores] = ga(problem);
elseif strcmp(optionsOuter.solver, 'ff')
    % Generate full factorial
    dFF = fullfact(optionsOuter.maxAgents*ones(1,nArchis));
    % Keep only groups below maxAgents agents
    indices = sum(dFF,2) <= optionsOuter.maxAgents;
    dFF = dFF(indices,:);

    % Evaluate fitness over full factorial
```

```

n = size(dFF,1);
fitness = zeros(n,1);

startTime = tic; % Start timer
if ~strcmp(optionsOuter.Display,'off')
    fprintf('Started on %s\n', datetime('now'))
end

for i = 1:n
    % Run inner loop
    fitness(i) = optionsOuter.fitnessFcn(dFF(i,:), optionsInner);

    % Estimate remaining time
    elapsed = toc(startTime);
    total = n*elapsed/i;
    remaining = total - elapsed;
    if strcmp(optionsOuter.Display,'iter')
        fprintf('[%03.0f%% (%d/%d)] (elapsed: %fs, remaining: %fs,
total: %fs)\n', 100*i/n, i, n, elapsed, remaining, total)
    end
end

% Display
if ~strcmp(optionsOuter.Display,'off')
    fprintf('\nFinished on %s\n', datetime('now'));
    fprintf('\tIterations: %d\n', n)
    fprintf('\tFunEval:    %d\n', n)
    fprintf('\tTotal time: %fs (%fs/iteration)\n', toc(startTime),
toc(startTime)/n)
end

% Take best architecture
[M,I] = min(fitness);
x = dFF(I,:);
fval = M;
exitflag = 1;
output = [];
population = dFF;
scores = fitness;
else
    error('Solver argument to outerLoop must be either ''ga'' or
''ff''.')
end
end

```

loopInner2.m

A modified version of the inner loop of the optimization algorithm to be used in the developing phase.

```

function [fval,x,exitflag,output,population,scores] =
loopInner2(xOuter, optionsInner)
global elite
% Vector storing architectures in use
archis = 1:length(xOuter);

% Remove absent architectures from the group vectors
toKeep = xOuter > 0;
nVars = optionsInner.nVars;
nVars = nVars(toKeep);
optionsInner.nVars = nVars;
optionsInner.lb = optionsInner.lb(toKeep);
optionsInner.ub = optionsInner.ub(toKeep);
archis = archis(toKeep);
xOuter = xOuter(toKeep);

% Compute number of variables depending on heterogeneity level
if optionsInner.trueHeterogeneity
    nVarsTotalArchis = xOuter.*nVars; % Total variables per
architecture
else
    nVarsTotalArchis = nVars;
end
nVarsTotal = sum(nVarsTotalArchis);

% Compute start indices for each architecture
nArchis = length(xOuter);
startArchis = ones(1,nArchis);
for i = 2:nArchis
    startArchis(i) = startArchis(i-1) + nVarsTotalArchis(i-1);
end

% Compute bounds for the variables
[lb, ub] = createBounds(xOuter, optionsInner.lb, optionsInner.ub,
nVarsTotal, optionsInner.nVars, optionsInner.trueHeterogeneity);

% Compute range for the initial population
optionsInner.PopInitRange = [lb; ub];

% Convey variables to the output function
if optionsInner.elitism == true
    optionsInner.OutputFcns =
{@(options, state, flag) gaInnerLoopElitism(options, state, flag, xOuter, nVar
s, archis, optionsInner.eliteFraction, startArchis)};
end

% TODO REMOVE HACK
% optionsInner.OutputFcns =
{@(options, state, flag) gaInnerLoopElitism(options, state, flag, xOuter, nVar
s, archis, optionsInner.eliteFraction, startArchis)};
% TODO REMOVE HACK - Compute the expected optimum
% offsetsArchis = 1./(1 + (xOuter - archis).^2);
% optionsInner.FitnessLimit = -.99*sum(offsetsArchis);

% If all elite cells are filled

```

```

if optionsInner.elitism == true && sum(cellfun(@isempty,elite)) == 0
    % Initialize population to elite
    optionsInner.InitialPopulation =
generateInitialPopulation(xOuter,elite,nVars,archis,startArchis,nVarsTotal,
optionsInner.trueHeterogeneity);
end

% Define optimization problem
problem = struct;
problem.fitnessfcn = @(x)optionsInner.fitnessFcn(x, xOuter,
optionsInner.nVars, archis, optionsInner.trueHeterogeneity);
problem.nvars = nVarsTotal;
problem.Aineq = [];
problem.Bineq = [];
problem.Aeq = [];
problem.beq = [];
problem.lb = lb;
problem.ub = ub;
problem.nonlcon = [];
problem.intcon = [];
problem.solver = 'ga';
problem.options = optionsInner;

% Optimize
[x,fval,exitflag,output,population,scores] = ga(problem);

% TODO remove this
% fprintf('Status: %d (%d generations)\n',exitflag, output.generations)
end

function [lb, ub] = createBounds(xOuter, lbArchis, ubArchis,
nVarsTotal, nVars, trueHeterogeneity)
% Initialization
lb = zeros(1, nVarsTotal);
ub = zeros(1, nVarsTotal);

% For each architecture
idx = 1;
for i = 1:length(xOuter)
    % Number of agents for this architecture
    nAgents = xOuter(i);
    % Number of variables for this architecture
    nv = nVars(i);
    % For each variable the architecture
    for k = 1:nv
        if trueHeterogeneity
            % For each agent with this architecture
            for j = 1:nAgents
                % Get upper and lower bounds
                lb(idx) = lbArchis{i}(k);
                ub(idx) = ubArchis{i}(k);

                % Increment array index
                idx = idx + 1;
            end
        else

```



```

        % Get upper and lower bounds
        lb(idx) = lbArchis{i}(k);
        ub(idx) = ubArchis{i}(k);

        % Increment array index
        idx = idx + 1;
    end
end
end

function population =
generateInitialPopulation(xOuter,elite,nVars,archis,startArchis,nVarsTotal,trueHeterogeneity)
% Initialization
population = zeros(size(elite{1}.scores,1), nVarsTotal);
nArchis = length(xOuter);

if trueHeterogeneity == true
    % For each architecture
    for i = 1:nArchis
        % For each agent with this architecture
        for j = 1:xOuter(i)
            startIndex = startArchis(i) + (j-1)*nVars(i);
            endIndex = startIndex + nVars(i) - 1;
            population(:,startIndex:endIndex) =
elite{archis(i)}.population;
        end
    end
else
    % For each architecture
    for i = 1:nArchis
        startIndex = startArchis(i);
        endIndex = startIndex + nVars(i) - 1;
        population(:,startIndex:endIndex) =
elite{archis(i)}.population;
    end
end
end
end

```

B.2.5 Plots optimizer

The files used to generate the plots presented in section 5.2.3 (see page 376).

runReplication.m

A script setting up a proper optimization problem with the test function and carrying out the optimization. The results are saved in a folder under a *replication.mat* file.

```

function runReplication(folderName, elitism, eliteFraction)
% Prepare workspace
clc
close all
clear global
clearvars -except folderName elitism eliteFraction

addpath('../././models')
addpath('../././optimizer')

global generationsArray generationOuter
generationsArray = {};
generationOuter = 1;

% Outer loop
=====
nArchis = 3;
maxAgents = 6;
nVars = [2 3 1];
solver = 'ga';
trueHeterogeneity = false;

optionsOuter = gaoptimset(...
    'Display', 'none',...
    'OutputFcns', {@gaOuterLoopElitism, @gaEstimateTime},...
    'PlotFcn', {@gaplotbestf},...
    'PlotInterval', 1,...
    'PopulationSize', 50,...
    'StallGenLimit', 20,...
    'TolFun', 1e-6,...
    'UseParallel', false,...
    'Vectorized', 'off');

optionsOuter.nArchis = nArchis; % Number of architectures
optionsOuter.maxAgents = maxAgents; % Maximum number of agents in the
group
optionsOuter.solver = solver;
optionsOuter.fitnessFcn = @loopInner;
optionsOuter.elitism = elitism;

% Inner loop
=====
optionsInner = gaoptimset(...
    'TolCon', 1e-3,...
    'Display', 'none',...
    'PlotFcn', {},...
    'PlotInterval', 1,...
    'PopulationSize', 200,...
    'StallGenLimit', 20,...
    'TolFun', 1e-6,...
    'UseParallel', false);

% Custom options
optionsInner.fitnessFcn = @testFitnessFunctionInnerLoop;

```

```

optionsInner.trueHeterogeneity = trueHeterogeneity; % Regroup per
architecture or not
optionsInner.nVars = nVars; % Number of variables per architecture
optionsInner.eliteFraction = eliteFraction;

% Bounds
for i = 1:length(nVars)
    % Lower bounds
    optionsInner.lb{i} = zeros(1, nVars(i));
    % Upper bounds
    optionsInner.ub{i} = (nArchis + 3)*ones(1, nVars(i));
end

% Optimize
=====
[x,fval,exitflag,output,population,scores] = loopOuter(optionsOuter,
optionsInner);

% Save results
=====
% Find unique file name
i = 1;
filename = sprintf(strcat(folderName, '/replication_%04d.mat'),i);
while i < 10000 && exist(filename, 'file') == 2
    filename = sprintf(strcat(folderName, '/replication_%04d.mat'),i);
    i = i+1;
end
save(filename)
end

```

plotGAvsFullFactorial.m

Comparison between the full factorial approach and the genetic algorithm approach for the outer loop optimizer.

```

% Prepare workspace
clc
close all
clear global
clear

addpath('../models')
addpath('../optimizer')

% Variables
nMax = 10; % Maximum number of agents per architecture
nArchis = 1:10; % Number of architectures

%% Assess function calls for the full factorial
fullFact = nMax.^nArchis; % Number of function calls with
a full factorial DOE

```

```

nMaxVec = [2:9,10:10:100];
figure(1)
% c = [winter(length(2:9)); autumn(length(10:10:100))];
my_legend = cell(1, length(nMaxVec)+1);
c = hsv(length(nMaxVec));
for i = 1:length(nMaxVec)
    fullFact = nMaxVec(i).^nArchis;
    semilogy(nArchis, fullFact, 'Color',c(i,:))
    my_legend{i} = sprintf('Full factorial (N_{max} = %d)',nMaxVec(i));
    hold on
end
hold off

%% Assess function calls for the GA
% populations = 50; % Population size for the GA
% generations = 50 + 10*nArchis; % Approximation based on observations
% ga = generations.*populations; % Estimated number of
function calls for the GA

% Outer loop
optionsOuter = gaoptimset(...
    'Display', 'off',...
    'OutputFcns', {},...
    'PlotFcn', {},...
    'PlotInterval', 1,...
    'PopulationSize', 50,...
    'TolFun', 1e-6,...
    'UseParallel', false);

optionsOuter.maxAgents = nMax; % Maximum number of agents in the group
optionsOuter.solver = 'ga';
optionsOuter.fitnessFcn = @testFitnessFunctionOuterLoop;
optionsOuter.elitism = false;

% Inner loop
optionsInner = [];

ga = zeros(1,length(nArchis));
for i = 1:length(nArchis)
    n = nArchis(i);

    % Compute full factorial case
    % dFF0 = fullfact((nMax+1)*ones(1,n))-1;
    % indices = sum(dFF0,2) <= nMax;
    % dFF = dFF0(indices,:);
    % fullFact(i) = size(dFF,1);

    % Compute GA case
    optionsOuter.nArchis = n; % Number of architectures
    for rep = 1:10
        [~,~,~,output,~,~] = loopOuter(optionsOuter, optionsInner);
        ga(rep,i) = output.funccount;
    end
end
end

```

```

ga = mean(ga);

% Plot
figure(1)
hold on
semilogy(nArchis, ga, 'k', 'LineWidth', 1)
hold off

my_legend{end} = 'GA';

%% Finish plot
xlabel('Number of architectures', 'FontName','Times New Roman',
'FontSize',25)
ylabel('Function calls', 'FontName','Times New Roman', 'FontSize',25)
set(gca, 'FontName', 'Times New Roman', 'FontSize',25)
h = legend(my_legend, 'location', 'eastoutside');
set(h, 'FontName', 'Times New Roman', 'FontSize',17)

```

analyzeResults.m

Analyzes the optimization results given a folder name that contains the MAT-files for a certain number of replications. All the files in the folder must stem from the same optimizer settings for the results to make sense. In particular, it computes the average and standard deviation of the number of inner loop generations, as well as their extreme values.

```

function [genAvg, genStd, lower, higher] = analyzeResults(folderName)
% Count number of files available
nFiles = length(dir(folderName)) - 2;

% Count maximum number of generations
genMax = 0;
for k = 1:nFiles
    % Load the results
    load(sprintf(strcat(folderName, '/replication_%04d.mat'), k))
    if size(generationsArray,1) > genMax
        genMax = size(generationsArray,1);
    end
end

% Initialize storing structures
generationsInner = cell(1, genMax);

% Accumulate measurements from all replications
for k = 1:nFiles
    % Load the results
    load(sprintf(strcat(folderName, '/replication_%04d.mat'), k))
    % Store results

```

```

    genMaxRep = size(generationsArray,1);
    for j = 1:genMaxRep
        generationsInner{j} = [generationsInner{j},
generationsArray{j}];
    end
end

% Statistical analysis for each generation of the outer loop
genAvg = zeros(1,genMax);
genStd = zeros(1,genMax);
higher = zeros(1,genMax);
lower = zeros(1,genMax);
for k = 1:genMax
    genAvg(k) = mean(generationsInner{k}); % Average
    genStd(k) = std(generationsInner{k}); % Standard deviation
    higher(k) = prctile(generationsInner{k},97.5); % Percentile
    lower(k) = prctile(generationsInner{k},2.5); % Percentile
end
end

```

analyzeResultsOuterLoop.m

A histogram plot of the average number of generations of the outer loop.

```

function [genAvg, genStd, genLower, genHigher,...
    funcAvg, funcStd, funcLower, funcHigher,...
    generationsOuter,funcCountOuter] =
analyzeResultsOuterLoop(folderName)
% Count number of files available
nFiles = length(dir(folderName)) - 2;

% Count maximum number of generations
generationsOuter = zeros(1,nFiles);
funcCountOuter = zeros(1,nFiles);
for k = 1:nFiles
    % Load the results
    load(sprintf(strcat(folderName,'/replication_%04d.mat'),k))
    generationsOuter(k) = output.generations;
    funcCountOuter(k) = output.funccount;
end

% Statistical analysis
genAvg = mean(generationsOuter); % Average
genStd = std(generationsOuter); % Standard deviation
genHigher = prctile(generationsOuter,97.5); % Percentile
genLower = prctile(generationsOuter,2.5); % Percentile

funcAvg = mean(funcCountOuter); % Average
funcStd = std(funcCountOuter); % Standard deviation
funcHigher = prctile(funcCountOuter,97.5); % Percentile
funcLower = prctile(funcCountOuter,2.5); % Percentile

% Plots

```

```

figure
hist(generationsOuter)
xlabel('Number of outer loop generations at convergence')
ylabel('Occurences')
title(folderName)
end

```

plotCompareElitismFraction.m

A comparison of the algorithm performance for different rates of elitism retention.

```

% Prepare workspace
clc
close all
clear global
clear

% Get statistical analysis
genAvg = cell(1,5);
lower = cell(1,5);
higher = cell(1,5);
for i = 1:5
    fprintf('Case %d/5\n', i)
    folderName = sprintf('results_Elitism%03dPureHeterogeneity', (i-
1)*25);
    [a, ~, l, h] = analyzeResults(folderName);
    genAvg{i} = a;
    lower{i} = l;
    higher{i} = h;
end

%% Plot
gray0 = 0.5*ones(3,1);
c = get(groot, 'DefaultAxesColorOrder');

figure
hold on

hPlain = plot(genAvg{1}, 'LineWidth',2, 'Color', gray0);
% hDashed = plot(lower{1}, '--', 'Color', gray0);
% hEmpty = plot(lower{1}, 'w');

h = zeros(1,5);
for i = 1:5
    h(i) = plot(genAvg{i}, 'LineWidth',2, 'Color', c(i,:));
    % plot(lower{i}, '--', 'Color', c(i,:));
    % plot(higher{i}, '--', 'Color', c(i,:));
end

hold off

```

```

xlabel('Outer loop generations','FontName','Times New
Roman','FontSize',22)
ylabel('Average inner loop generations required','FontName','Times New
Roman','FontSize',22)
ylabel({'Average inner loop';'generations required'},'FontName','Times
New Roman','FontSize',22)
% hLegend = legend([h,hEmpty,hPlain,hDashed],...
%     {'\kappa = 0%',...
%     '\kappa = 25%',...
%     '\kappa = 50%',...
%     '\kappa = 75%',...
%     '\kappa = 100%',...
%     '',...
%     'Average',...
%     '95% confidence interval'});
hLegend = legend([h],...
    {'\kappa = 0%',...
    '\kappa = 25%',...
    '\kappa = 50%',...
    '\kappa = 75%',...
    '\kappa = 100%'});
set(hLegend,'FontName','Times New Roman','FontSize',22)
set(gca,'FontName','Times New Roman','FontSize',22)

```

plotCompareElitismTrueFalse.m

A comparison of the performance of the algorithm when the elitism is activated or not.

```

% Prepare workspace
clc
close all
clear global
clear

% Get statistical analysis
folderName = 'results_NoElitismPureHeterogeneity';
% folderName = 'results_NoElitismPartialHeterogeneity';
% folderName = 'results_Elitism000PartialHeterogeneity'
[genAvg0, ~, lower0, higher0] = analyzeResults(folderName);

folderName = 'results_WithElitism05PureHeterogeneity';
% folderName = 'results_WithElitism05PartialHeterogeneity';
[genAvg1, ~, lower1, higher1] = analyzeResults(folderName);

%% Plot
gray0 = 0.5*ones(3,1);
figure
hold on
h0 = plot(genAvg0, 'LineWidth',2,'Color',gray0);
h2 = plot(genAvg0,'LineWidth',2,'Color',gray0);
h3 = plot(lower0,'--','Color',gray0);
plot(higher0,'--','Color',gray0)
h1 = plot(genAvg1,'b','LineWidth',2);

```



```

hEmpty = plot(lower1, '--w');

plot(lower1, '--b')
plot(higher1, '--b')
hold off
xlabel('Outer loop generations','FontName','Times New
Roman','FontSize',12)
ylabel('Average inner loop generations required','FontName','Times New
Roman','FontSize',12)
h = legend([h0,h1,hEmpty,h2,h3],{'Without elitism','Elitism
50%', ' ','Average','95% confidence interval'});
set(h, 'FontName', 'Times New Roman', 'FontSize',12)
set(gca, 'FontName', 'Times New Roman', 'FontSize',12)

ylim([0, 220])

% Plot 2 -----
---
figure
hold on

hPlain = plot(genAvg0, 'LineWidth',2, 'Color',gray0);
hDashed = plot(lower1, '--', 'Color',gray0);
hEmpty = plot(lower1, 'Color', 'none');

h0 = plot(genAvg0, 'LineWidth',2, 'Color','r');
plot(lower0, '--', 'Color', [255, 183, 183]/255);
plot(higher0, '--', 'Color', [255, 183, 183]/255);

h1 = plot(genAvg1, 'b', 'LineWidth',2);
plot(lower1, '--', 'Color', [206, 191, 255]/255)
plot(higher1, '--', 'Color', [206, 191, 255]/255)

hold off

xlabel('Outer loop generations','FontName','Times New
Roman','FontSize',12)
ylabel('Average inner loop generations required','FontName','Times New
Roman','FontSize',12)
h = legend([h0,h1,hEmpty,hPlain,hDashed],{'Without elitism','Elitism
50%', ' ','Average','95% confidence interval'});
set(h, 'FontName', 'Times New Roman', 'FontSize',12)
set(gca, 'FontName', 'Times New Roman', 'FontSize',12)

% Transparent background
set(gcf, 'Color', 'None')
set(gca, 'Color', 'None', 'XColor', 'w', 'YColor', 'w', 'ZColor', 'w')
set(h, 'color', 'none', 'TextColor', 'w', 'EdgeColor', 'w')

ylim([0, 220])

```

plotCompareHeterogeneityFraction.m

A comparison of the different elitism rates when heterogeneity is partial only.

```

% Prepare workspace
clc
close all
clear global
clear

% Get statistical analysis
genAvg = cell(1,5);
lower = cell(1,5);
higher = cell(1,5);
for i = 1:5
    fprintf('Case %d/5\n', i)
    folderName = sprintf('results_Elitism%03dPartialHeterogeneity', (i-
1)*25);
    [a, ~, l, h] = analyzeResults(folderName);
    genAvg{i} = a;
    lower{i} = l;
    higher{i} = h;
end

%% Plot
gray0 = 0.5*ones(3,1);
c = get(groot, 'DefaultAxesColorOrder');

figure
hold on

hPlain = plot(genAvg{1}, 'LineWidth',2, 'Color',gray0);
hDashed = plot(lower{1}, '--', 'Color',gray0);
hEmpty = plot(lower{1}, 'w');

h = zeros(1,5);
for i = 1:5
    h(i) = plot(genAvg{i}, 'LineWidth',2, 'Color',c(i,:));
    plot(lower{i}, '--', 'Color',c(i,:));
    plot(higher{i}, '--', 'Color',c(i,:));
end

hold off

xlabel('Outer loop generations', 'FontName', 'Times New
Roman', 'FontSize',12)
ylabel('Average inner loop generations required', 'FontName', 'Times New
Roman', 'FontSize',12)
% ylabel({'Average inner loop'; 'generations
required'}, 'FontName', 'Times New Roman', 'FontSize',22)
hLegend = legend([h,hEmpty,hPlain,hDashed],...
    {'\kappa = 0%',...
    '\kappa = 25%',...
    '\kappa = 50%',...
    '\kappa = 75%',...
    '\kappa = 100%',...
    '' ,...

```

```

    'Average',...
    '95% confidence interval'});
% hLegend = legend([h],...
%     {'\kappa = 0%',...
%     '\kappa = 25%',...
%     '\kappa = 50%',...
%     '\kappa = 75%',...
%     '\kappa = 100%'});
set(hLegend,'FontName','Times New Roman','FontSize',12)
set(gca,'FontName','Times New Roman','FontSize',12)

```

plotCompareHeterogeneityTrueFalse.m

A plot comparing the performance of the algorithm between elitism and non-elitism, and between partial and full heterogeneity.

```

% Prepare workspace
clc
close all
clear global
clear

% Get statistical analysis

% Full heterogeneity
folderName = 'results_NoElitismPureHeterogeneity';
[genAvg0, ~, lower0, higher0] = analyzeResults(folderName);

folderName = 'results_WithElitism05PureHeterogeneity';
[genAvg1, ~, lower1, higher1] = analyzeResults(folderName);

% Partial heterogeneity
folderName = 'results_NoElitismPartialHeterogeneity';
[genAvg2, ~, lower2, higher2] = analyzeResults(folderName);

folderName = 'results_WithElitism05PartialHeterogeneity';
[genAvg3, ~, lower3, higher3] = analyzeResults(folderName);

%% Plot
gray0 = 0.5*ones(3,1);

figure
hold on

hPlain = plot(genAvg0,'LineWidth',2,'Color',gray0);
hDashed = plot(lower3,'--','Color',gray0);
hEmpty = plot(lower3,'--w','LineWidth',2);

h0 = plot(genAvg0, 'LineWidth',2,'Color','r');
% plot(lower0,'--','Color',[255, 183, 183]/255)

```

```

% plot(higher0,'--','Color',[255, 183, 183]/255)

h1 = plot(genAvg1,'b','LineWidth',2);
% plot(lower1,'--','Color',[206, 191, 255]/255)
% plot(higher1,'--','Color',[206, 191, 255]/255)

h2 = plot(genAvg2,'m','LineWidth',2);
plot(lower2,'--','Color',[255, 130, 220]/255)
plot(higher2,'--','Color',[255, 130, 220]/255)

h3 = plot(genAvg3,'c','LineWidth',2);
plot(lower3,'--','Color',[130, 249, 255]/255)
plot(higher3,'--','Color',[130, 249, 255]/255)

hold off

xlabel('Outer loop generations','FontName','Times New
Roman','FontSize',12)
ylabel('Average inner loop generations required','FontName','Times New
Roman','FontSize',12)
h = legend([h0,h1,h2,h3,hEmpty,hPlain,hDashed],...
{'Full heterogeneity - No elitism',...
'Full heterogeneity - Elitism 50%',...
'Partial heterogeneity - No elitism',...
'Partial heterogeneity - Elitism 50%',...
'',...
'Average',...
'95% confidence interval'});
set(h,'FontName','Times New Roman','FontSize',10)
set(gca,'FontName','Times New Roman','FontSize',12)

ylim([0, 220])

```

plotElitism.m

A simple plot of the influence of elitism on the algorithm.

```

% Prepare workspace
clc
% close all
clear global
clear

folderName = 'results_Elitism01PureHeterogeneity';
folderName = 'results_Elitism05PureHeterogeneity';

% Count number of files available
nFiles = length(dir(folderName)) - 2;

% Count maximum number of generations
genMax = 0;
for i = 1:nFiles

```

```

    % Load the results
    load(sprintf(strcat(folderName, '/replication_%04d.mat'), i))
    if size(GenerationsArray, 1) > genMax
        genMax = size(GenerationsArray, 1);
    end
end

% Initialize storing structures
GenerationsInner = cell(1, genMax);

% Accumulate measurements from all replications
for i = 1:nFiles
    % Load the results
    load(sprintf(strcat(folderName, '/replication_%04d.mat'), i))
    % Store results
    genMaxRep = size(GenerationsArray, 1);
    for j = 1:genMaxRep
        GenerationsInner{j} = [GenerationsInner{j},
GenerationsArray{j}];
    end
end

% Statistical analysis for each generation of the outer loop
genAvg = zeros(1, genMax);
genStd = zeros(1, genMax);
higher = zeros(1, genMax);
lower = zeros(1, genMax);
for i = 1:genMax
    genAvg(i) = mean(GenerationsInner{i}); % Average
    genStd(i) = std(GenerationsInner{i}); % Standard deviation
    higher(i) = prctile(GenerationsInner{i}, 97.5); % Percentile
    lower(i) = prctile(GenerationsInner{i}, 2.5); % Percentile
end

% % Remove NaN
% genAvg(isnan(genAvg)) = [];
% genStd(isnan(genStd)) = [];

% Plot
% figure
hold on
plot(genAvg, 'LineWidth', 2)
% plot(genAvg + 3*genStd, '--r')
% plot(genAvg - 3*genStd, '--r')
plot(higher, '--m')
plot(lower, '--m')
hold off
xlabel('Outer loop generations')
ylabel('Average inner loop generations required')
legend('Average', '95% confidence interval')
ylim([0 150])

```

plotElitismImprovement.m

A plot of the performance of the algorithm as a function of the elitism rate.

```
% Prepare workspace
clc
close all
clear global
clear

% Data
kappa = 0:0.25:1.0;
maxGen = [303, 133, 137, 92, 43];
gen3 = [37, 16, 15, 14, 13];

%% Plot 1
figure
plot(kappa,maxGen)
hold on
plot(kappa,maxGen, '*')
hold off
xlabel('\kappa', 'FontName', 'Times New Roman', 'FontSize', 20)
ylabel({'Maximum number'; 'of'; 'generations observed'}, 'FontName', 'Times
New Roman', 'FontSize', 20)
set(gca, 'FontName', 'Times New Roman', 'FontSize', 20)

%% Plot 2
figure
plot(kappa,gen3)
hold on
plot(kappa,gen3, '*')
hold off
xlabel('\kappa', 'FontName', 'Times New Roman', 'FontSize', 20)
ylabel({'Number of inner generations'; 'at 3rd outer
generation'}, 'FontName', 'Times New Roman', 'FontSize', 20)
set(gca, 'FontName', 'Times New Roman', 'FontSize', 20)
```

plotOuterCompareHeterogeneityFraction.m

A display of the performance of the outer loop depending on the elitism rate.

```
% Prepare workspace
clc
close all
clear global
clear

% Full heterogeneity
genAvg = zeros(1,5);
genLower = zeros(1,5);
genHigher = zeros(1,5);
genAvgPartial = zeros(1,5);
genLowerPartial = zeros(1,5);
genHigherPartial = zeros(1,5);
```

```

for i = 1:5
    fprintf('Case %d/5\n',i)
    folderName = sprintf('results_Elitism%03dPureHeterogeneity', (i-
1)*25);
    [genAvg0, genStd0, genLower0, genHigher0, funcAvg0, funcStd0,
funcLower0, funcHigher0, generationsOuter0, funcCountOuter0] =
analyzeResultsOuterLoop(folderName);
    genAvg(i) = genAvg0;
    genLower(i) = genLower0;
    genHigher(i) = genHigher0;

    folderName = sprintf('results_Elitism%03dPartialHeterogeneity', (i-
1)*25);
    [genAvg0, genStd0, genLower0, genHigher0, funcAvg0, funcStd0,
funcLower0, funcHigher0, generationsOuter0, funcCountOuter0] =
analyzeResultsOuterLoop(folderName);
    genAvgPartial(i) = genAvg0;
    genLowerPartial(i) = genLower0;
    genHigherPartial(i) = genHigher0;
end

%% Plot
gray0 = 0.5*ones(3,1);
c = get(groot, 'DefaultAxesColorOrder');

figure
hold on
bar(.75, genAvg(1), 0.4, 'FaceColor', gray0)
bar(.75, genAvg(1), 0.4, 'FaceColor', gray0, 'LineStyle', '--
', 'FaceAlpha', 0.5)
bar(.75, genAvg(1), 0.4, 'FaceColor', gray0, 'LineStyle', 'none', 'FaceAlpha',
0)

for i = 1:5
    bar(i-0.25, genAvg(i), 0.4, 'FaceColor', c(i,:))
end

for i = 1:5
    bar(i + 0.25, genAvgPartial(i), 0.4, 'FaceColor', c(i,:), 'LineStyle', '-
-', 'FaceAlpha', 0.5)
end

hLegend = legend(...
    'Full heterogeneity', ...
    'Partial heterogeneity', ...
    '', ...
    '\kappa = 0%', ...
    '\kappa = 25%', ...
    '\kappa = 50%', ...
    '\kappa = 75%', ...
    '\kappa = 100%');
errorbar([1 2 3 4 5] - 0.25, ...
    genAvg, ...
    genAvg - genLower, ...
    genHigher - genAvg, ...
    'lineStyle', 'none', ...

```

```

        'Color','k')
errorbar([1 2 3 4 5] + 0.25,...
        genAvgPartial,...
        genAvgPartial - genLowerPartial,...
        genHigherPartial - genAvgPartial,...
        'LineStyle','none',...
        'Color','k')
hold off

set(gca,'XTickLabel','')
ylabel({'Outer loop ','generations required'},'FontName','Times New
Roman','FontSize',12)

set(hLegend,'FontName','Times New Roman','FontSize',10)
set(gca,'FontName','Times New Roman','FontSize',12)

```

plotOuterCompareHeterogeneityTrueFalse.m

Analysis of the outer loop for various conditions of heterogeneity and elitism.

```

% Prepare workspace
clc
close all
clear global
clear

% Full heterogeneity
fprintf('Case 1/4\n')
folderName = 'results_NoElitismPureHeterogeneity';
[genAvg0, genStd0, genLower0, genHigher0, funcAvg0, funcStd0,
funcLower0, funcHigher0, generationsOuter0,funcCountOuter0] =
analyzeResultsOuterLoop(folderName);

fprintf('Case 2/4\n')
folderName = 'results_WithElitism05PureHeterogeneity';
[genAvg1, genStd1, genLower1, genHigher1, funcAvg1, funcStd1,
funcLower1, funcHigher1, generationsOuter1,funcCountOuter1] =
analyzeResultsOuterLoop(folderName);

% Partial heterogeneity
fprintf('Case 3/4\n')
folderName = 'results_NoElitismPartialHeterogeneity';
[genAvg2, genStd2, genLower2, genHigher2, funcAvg2, funcStd2,
funcLower2, funcHigher2, generationsOuter2,funcCountOuter2] =
analyzeResultsOuterLoop(folderName);

fprintf('Case 4/4\n')
folderName = 'results_WithElitism05PartialHeterogeneity';
[genAvg3, genStd3, genLower3, genHigher3, funcAvg3, funcStd3,
funcLower3, funcHigher3, generationsOuter3,funcCountOuter3] =
analyzeResultsOuterLoop(folderName);

```



```

%% Plots
figure
hold on
bar(1,genAvg0,'r')
bar(2,genAvg1,'b')
bar(3,genAvg2,'m')
bar(4,genAvg3,'c')
hLegend = legend('Full heterogeneity - No elitism',...
    'Full heterogeneity - Elitism 50%',...
    'Partial heterogeneity - No elitism',...
    'Partial heterogeneity - Elitism 50%','location','northoutside');

genAvg = [genAvg0, genAvg1, genAvg2, genAvg3];
genLower = genAvg - [genLower0, genLower1, genLower2, genLower3];
genHigher = [genHigher0, genHigher1, genHigher2, genHigher3] - genAvg;
errorbar([1 2 3 4],...
    genAvg,...
    genLower,...
    genHigher,...
    'lineStyle','none',...
    'Color','k')
hold off

set(gca,'XTickLabel','')
ylabel({'Outer loop ','generations required'},'FontName','Times New Roman','FontSize',12)

set(hLegend,'FontName','Times New Roman','FontSize',10)
set(gca,'FontName','Times New Roman','FontSize',12)

```

B.3 *Design space exploration*

This section contains a Matlab implementation of the morphological tree framework which was introduced in section 5.1 page 292.

B.3.1 **Classes**

The files of the object-oriented morphological tree data structure. See section 5.1.3 page 301 for a complete description of the class architecture.

option.m

```

classdef option
    %OPTION Represents a morphological option to be used in a row of a
    %morphological matrix
    % Detailed explanation goes here

```

```

properties (Access = private)
    index = 0;
    name = '';
    variables = {};
end

methods

function obj = option(name,variables)
    obj.name = name;
    obj.variables = variables;
end

function n = countVariables(obj)
    n = length(obj.variables);
end

function s = toString(obj)
    % Prepare variables string
    str = '';
    vars = obj.variables;
    for j = 1:length(vars)
        str = strcat(str,vars{j},',');
    end
    % Remove last comma
    str = str(1:end-1);

    % Add to variable name
    s = ['(',num2str(obj.index),') ',obj.name,' {' ,str,'}'];
end

% Getters
function out = getName(obj)
    out = obj.name;
end

function out = getIndex(obj)
    out = obj.index;
end

function out = getVariables(obj)
    out = obj.variables;
end

% Setter
function obj = setIndex(obj,index)
    obj.index = index;
end

end
end

```

row.m

```
classdef (Abstract) row
```

```

%ROW Summary of this class goes here
% Detailed explanation goes here
properties (Access = protected)
    name = '';
    options = {};
end

methods
    % Methods
    function obj = addOption(obj,option)
        n = size(obj.options,1);
        obj.options{n+1,1} = option;
    end

    function [obj, bool] = removeOption(obj,name)
        % Initialization
        i = 1;
        % Find the option (it is assumed there is only one option
with
        % this name)
        while i < size(obj.options,1) &&
~strcmp(obj.options{i,1}.getName,name)
            i = i + 1;
        end

        % If option was found
        if strcmp(obj.options{i,1}.getName,name)
            % Remove option
            obj.options(i,:) = [];
            bool = true;
        else
            bool = false;
        end
    end

    function obj = addOptionFromMorph(obj,name,m)
        % NOTE: the morph matrix must be single level
        % Initialization
        rows = m.getRows;

        % Save number of options for each row
        m = size(rows,1);
        nOptions = zeros(1,m);
        for i = 1:m
            nOptions(i) = size(rows{i}.options,1);
        end

        % Assemble each possible choice
        combinations = fullfact(nOptions); % Each row is a
combination

        % For each combination
        for i = 1:size(combinations,1)
            % Initialization
            nameSubOption = strcat(name, ' (');

```

```

        variablesSubOption = {};
        counter = 1;
        indices = combinations(i,:);
        % For each row of the morphological matrix
        for j = 1:m
            % Append names
            nameSubOption = sprintf('%s%s, ',
nameSubOption,rows{j}.getOptions{indices(j)}.getName);
            % Append variables
            variablesTemp =
rows{j}.getOptions{indices(j)}.getVariables;
            for k = 1:length(variablesTemp)
                variablesSubOption{counter} = variablesTemp{k};
                counter = counter + 1;
            end
        end
        % Remove last coma and add closing bracket
        nameSubOption = nameSubOption(1:end-2);
        nameSubOption = strcat(nameSubOption,')');

        % Add option to row
        obj =
obj.addOption(option(nameSubOption,variablesSubOption));
    end
end

function [obj, bool, nVars, nOptions] = reduce(obj)
    % Initialization
    bool = false;
    nVars = 0; % Number of variables removed
    nOptions = 0; % Number of options removed
    n = size(obj.options,1);

    % For each option
    i = 1;
    while i < n+1
        % Initialization
        j = 1;
        vars = cell2mat(obj.options{i,1}.getVariables);

        % For all other options
        while j < n+1
            % Check if it has the same variables
            if i~=j &&
size(cell2mat(obj.options{j,1}.getVariables),1) > 0 &&...
strcmp(cell2mat(obj.options{j,1}.getVariables),vars)
                % Increment variables counter
                nVars = nVars +
length(obj.options{j,1}.getVariables);
                nOptions = nOptions + 1;
                % Remove option
                obj =
obj.removeOption(obj.options{j,1}.getName);
                bool = true;
                % Update size

```

```

        n = size(obj.options,1);
    else
        % Iterate
        j = j + 1;
    end
end

    % Iterate
    i = i + 1;
end
end

function n = countOptions(obj)
    n = size(obj.options,1);
end

function n = countVariables(obj)
    % Initialization
    n = 0;

    % Add variables
    for i = 1:size(obj.options,1)
        n = n + obj.options{i,1}.countVariables;
    end
end

function [obj, index] =
setUpCompatibilityIndices(obj, startIndex)
    % Initialization
    index = startIndex;

    % Set up indices for all options
    for i = 1:size(obj.options,1)
        obj.options{i,1} = obj.options{i,1}.setIndex(index);
        index = index + 1;
    end
end

function s = toString(obj)
    % Initialization
    s = strcat(obj.name, '\n');

    % Create string
    for i = 1:size(obj.options,1)
        s = strcat(s, '\t', obj.options{i}.toString, '\n');
    end
end

% Getters
function out = getName(obj)
    out = obj.name;
end

function out = getOptions(obj)
    out = obj.options;
end

```

```

        end
    end

    % Abstract methods
    methods (Abstract)
        computeAlternatives(obj)
    end
end

```

rowConventional.m

```

classdef rowConventional < row
    %ROW Summary of this class goes here
    % Detailed explanation goes here
    methods
        % Constructor
        function obj = rowConventional(name)
            obj.name = name;
        end

        % Methods
        function n = computeAlternatives(obj)
            n = size(obj.options,1);
        end
    end
end

```

rowCombinatorial.m

```

classdef rowCombinatorial < row
    %ROW Summary of this class goes here
    % Detailed explanation goes here
    methods
        function obj = rowCombinatorial(name)
            obj.name = name;
        end

        function nAlternatives = computeAlternatives(obj)
            % Initialization
            nAlternatives = 0;
            n = size(obj.options,1);

            % Possibility to choose from 1 to nOptions options in the
row
            for k = 1:n
                nAlternatives = nAlternatives + nchoosek(n,k);
            end
        end
    end
end

```

morphologicalMatrix.m

```
classdef morphologicalMatrix
    %MORPHOLOGICALMATRIX Summary of this class goes here
    % Detailed explanation goes here

    properties (Access = private)
        name = '';
        rows = {};
        abstract = false;
    end

    methods
        function obj = morphologicalMatrix(name, abstract)
            if nargin < 2
                abstract = false;
            end
            obj.name = name;
            obj.abstract = abstract;
        end

        function obj = addRow(obj,row)
            n = size(obj.rows,1);
            obj.rows{n+1,1} = row;
        end

        function [obj, bool] = removeRow(obj,name)
            % Initialization
            i = 1;
            % Find the row (it is assumed there is only one row with
            % this name)
            while i < size(obj.rows,1) &&
                ~strcmp(obj.rows{i,1}.getName,name)
                    i = i + 1;
            end

            % If option was found
            if strcmp(obj.rows{i,1}.getName,name)
                % Remove option
                obj.rows(i,:) = [];
                bool = true;
            else
                bool = false;
            end
        end

        function n = computeAlternatives(obj)
            % If empty matrix
            if 0 == size(obj.rows,1)
                n = 0;
                return
            end
            % Else
            n = 1;
            % Product of rows options
        end
    end
end
```

```

        for i = 1:length(obj.rows)
            n = n*obj.rows{i}.computeAlternatives;
        end
    end

function n = computeCompatibleAlternatives(obj,c,set,n)
    % If bottom of matrix is reached
    if size(obj.getRows,1) == 0
        % Increment counter of compatible alternatives
        n = n+1;
    else
        % Remove first row of the morphological matrix
        mReduced = obj.removeRow(obj.getRows{1,1}.getName);

        % Recursivity on all options of the first row
        options = obj.getRows{1,1}.getOptions;
        for i = 1:size(options,1)
            % TODO remove this debug
            if (strcmp(obj.name,'Fixed wing') ||
                (strcmp(obj.name,'Multirotor'))) && size(obj.getRows,1) == 20
                fprintf('%d/%d
                (%.2f%%)\n',i,size(options,1),100*i/size(options,1))
            end

            % Get option index
            idx = options{i,1}.getIndex;

            % Continue recursion only if option is compatible
            with current set
            if isempty(set) || (~isempty(set) &&
                prod(c(idx,set)) ~= 0)
                n =
                computeCompatibleAlternatives(mReduced,c,[set, idx],n);
            end
        end
    end
end

function [obj,bool,nVars,nOptions] = reduce(obj)
    % Initialization
    bool = false;
    nVars = 0;
    nOptions = 0;

    % For every row
    i = 1;
    while i < size(obj.rows,1)+1
        % Reduce it
        [temp, bool, removedVars, removedOptions] =
        obj.rows{i,1}.reduce;
        obj.rows{i,1} = temp;
        nVars = nVars + removedVars;
        nOptions = nOptions + removedOptions;

        % If it has only one option, remove it
    end
end

```



```

        if size(obj.rows{i,1}.getOptions,1) == 1
            obj = obj.removeRow(obj.rows{i,1}.getName);
        else
            i = i+1;
        end
    end
end

function n = countOptions(obj)
    % Initialization
    n = 0;

    % Add options
    for i = 1:size(obj.rows,1)
        n = n + obj.rows{i,1}.countOptions;
    end
end

function n = countVariables(obj)
    % Initialization
    n = 0;

    % Add options
    for i = 1:size(obj.rows,1)
        n = n + obj.rows{i,1}.countVariables;
    end
end

function obj = setUpCompatibilityIndices(obj)
    % Initialization
    index = 1;
    for i = 1:size(obj.rows,1)
        [temp, index] =
obj.rows{i,1}.setUpCompatibilityIndices(index);
        obj.rows{i,1} = temp;
    end
end

function c = initializeCompatibilityMatrix(obj)
    % Initialization: assume all compatible
    n = obj.countOptions;
    c = ones(n);

    % One option per row
    for i = 1:size(obj.rows,1)
        if isa(obj.rows{i,1}, 'rowConventional')
            options = obj.rows{i,1}.getOptions;
            n1 = options{1,1}.getIndex;
            n2 = options{end,1}.getIndex;
            for j = n1:n2-1
                for k = j+1:n2
                    c(j,k) = 0;
                end
            end
        end
    end
end

```

```

        end
    end

    function s = toString(obj)
        s = strcat(obj.name, '\n');
        for i = 1:length(obj.rows)
            s = strcat(s,obj.rows{i}.toString);
        end
    end

    % Getters
    function out = getName(obj)
        out = obj.name;
    end

    function out = getRows(obj)
        out = obj.rows;
    end

    function out = isAbstract(obj)
        out = obj.abstract;
    end

    % Setters
    function obj = setAbstract(obj, abstract)
        obj.abstract = abstract;
    end

    % Utility function
    function Asym = symmetrize(~,A)
        Adiag = diag(diag(A)); % Place diagonal elements
        Atri = triu(A,1); % Strictly upper matrix
        Asym = Adiag + Atri + Atri'; % Assemble final symmetric
matrix
    end
end
end
end

```

morphologicalTree.m

Uses the Matlab tree data structure proposed by [231].

```

classdef morphologicalTree < tree
    methods
        function obj = morphologicalTree(content)
            obj = obj@tree(content);
        end

        function n = computeAlternatives(obj)
            % Create a breadth first iterator
            iterator = obj.depthfirstiterator;
            n = 1;
        end
    end
end

```

```

        for it = iterator
            if ~obj.get(it).isAbstract()
                n = n*obj.get(it).computeAlternatives();
            end
        end
    end
end

function [obj,bool,nVars,nOptions] = reduce(obj)
    % Create a breadth first iterator
    bool = false;
    nVars = 0;
    nOptions = 0;
    iterator = obj.depthfirstiterator;
    for it = iterator
        [m,bool,n,no] = obj.get(it).reduce();
        obj = obj.set(it,m);
        nVars = nVars + n;
        nOptions = nOptions + no;
    end
end

function str = toString(obj)
    % Create duplicate representative tree
    tName = tree(obj,'clear');
    iterator = obj.breadthfirstiterator;
    for it = iterator
        if obj.get(it).isAbstract
            tName = tName.set(it, sprintf('[A] %s (%.4g)',
obj.get(it).getName(), obj.get(it).computeAlternatives()));
        else
            tName = tName.set(it, sprintf('%s (%.4g)',
obj.get(it).getName(), obj.get(it).computeAlternatives()));
        end
    end
    str = tName.toString;
end
end
end
end

```

B.3.2 Unit tests

test.m

The main file running all the unit tests of the test campaign.

```

% Prepare workspace
clc
close all
clear global
clear

% Run tests

```

```

result = runtests('testOption.m');
rt = table(result);
disp(rt)

result = runtests('testRow.m');
rt = table(result);
disp(rt)

result = runtests('testMorph.m');
rt = table(result);
disp(rt)

result = runtests('testTree.m');
rt = table(result);
disp(rt)

```

randomString.m

A function generating a random string given a string length. This is a utility function used to generate random option names in the unit tests.

```

function R = randomString(N)
SET = char(['a':'z' '0':'9']) ;
NSET = length(SET) ;

i = ceil(NSET*rand(1,N)) ; % with repeat
R = SET(i) ;
end

```

testOption.m

```

%% Main function to generate tests
function tests = testOption
tests = functiontests(localfunctions);
end

%% Test Functions
function testMain(~)
% Repeat random test
for i = 1:100
    % Generate random variables
    index = randi(100);
    name = randomString(randi(50));
    nVars = randi(100);
    variables = cell(1,nVars);
    for j = 1:nVars
        variables{j} = sprintf('x%d',j);
    end
end

```

```

    % Create random option
    o = option(name, variables);
    o = o.setIndex(index);

    % Test
    assert(strcmp(o.getName(), name))
    assert(o.getIndex() == index)
    assert(sum(ismember(o.getVariables, variables)) == nVars)
    assert(o.countVariables() == nVars)
end
end

%% Optional file fixtures
function setupOnce(~) % do not change function name
% Prepare workspace
close all
clear global
clear

addpath('../morphologicalTree')
addpath('utilities')
end

function teardownOnce(~) % do not change function name
% change back to original path, for example
end

%% Optional fresh fixtures
function setup(~) % do not change function name
end

function teardown(~) % do not change function name
end

```

testRow.m

```

%% Main function to generate tests
function tests = testRow
tests = functiontests(localfunctions);
end

%% Test Functions
function testConventional(~)
r = rowConventional('Feature 1');
r = r.addOption(option('Option 1', {'x1', 'x2', 'x3', 'x4'}));
r = r.addOption(option('Option 2', {'x1', 'x2', 'x3'}));
r = r.addOption(option('Option 3', {'x1', 'x2', 'x3'}));
r = r.addOption(option('Option 4', {'x5'}));

assert(strcmp(r.getName(), 'Feature 1'))
assert(r.countOptions() == 4)

```

```

assert(r.countVariables() == 11)
assert(r.computeAlternatives() == 4)

r = r.removeOption('X');
assert(r.countOptions() == 4)

r = r.removeOption('Option 4');
assert(r.countOptions() == 3)

r = r.reduce();
assert(r.countOptions() == 2)
end

function testCombinatorial(~)
r = rowCombinatorial('Feature 1');
r = r.addOption(option('Option 1', {'x1', 'x2', 'x3', 'x4'}));
r = r.addOption(option('Option 2', {'x1', 'x2', 'x3'}));
r = r.addOption(option('Option 3', {'x1', 'x2', 'x3'}));
r = r.addOption(option('Option 4', {'x5'}));

assert(strcmp(r.getName(), 'Feature 1'))
assert(r.countOptions() == 4)
assert(r.countVariables() == 11)
assert(r.computeAlternatives() == 15)

r = r.removeOption('X');
assert(r.countOptions() == 4)

r = r.removeOption('Option 4');
assert(r.countOptions() == 3)

r = r.reduce();
assert(r.countOptions() == 2)
end

%% Optional file fixtures
function setupOnce(~) % do not change function name
% Prepare workspace
close all
clear global
clear

addpath('../morphologicalTree')
addpath('utilities')
end

function teardownOnce(~) % do not change function name
% change back to original path, for example
end

%% Optional fresh fixtures
function setup(~) % do not change function name
end

```

```
function teardown(~) % do not change function name
end
```

testMorph.m

```
%% Main function to generate tests
function tests = testMorph
tests = functiontests(localfunctions);
end

%% Test Functions
function testMain(~)
% Initialize morphological matrix
m = morphologicalMatrix('Morphological Matrix');

% Add row -----
---
r = rowConventional('Feature 1');
r = r.addOption(option('Option 1', {'x1', 'x2', 'x3', 'x4'}));
r = r.addOption(option('Option 2', {'x1', 'x2', 'x3'}));
r = r.addOption(option('Option 3', {'x1', 'x2', 'x3'}));
r = r.addOption(option('Option 4', {'x5'}));
m = m.addRow(r); % Add row to matrix

% Add row -----
---
r = rowConventional('Feature 2');
r = r.addOption(option('Option 1', {'x1', 'x2', 'x3', 'x4'}));
r = r.addOption(option('Option 2', {'x1', 'x2', 'x3'}));
r = r.addOption(option('Option 3', {'x1', 'x2', 'x3'}));
m = m.addRow(r); % Add row to matrix

assert(m.computeAlternatives() == 12)

% Add row -----
---
r = rowCombinatorial('Feature 3 (Combinatorial)');
r = r.addOption(option('Option 1', {'x1', 'x2', 'x3', 'x4'}));
r = r.addOption(option('Option 2', {'x1', 'x2', 'x3'}));
r = r.removeOption('Option 2');
r = r.removeOption('Option 3');

r = r.addOption(option('Option 2', {'x1', 'x2', 'x3', 'x4'}));
r = r.addOption(option('Option 3', {'x1', 'x2', 'x3', 'x4'}));
r = r.addOption(option('Option 4', {'x1', 'x2', 'x3', 'x4'}));
r = r.addOption(option('Option 5', {'x1', 'x2', 'x3', 'x4'}));
r = r.addOption(option('Option 6', {'x1', 'x2', 'x3'}));
r = r.addOption(option('Option 7', {'x1', 'x2', 'x3'}));
r = r.addOption(option('Option 8', {'x1', 'x2', 'x3'}));

m = m.addRow(r); % Add row to matrix

assert(m.computeAlternatives() == 12*r.computeAlternatives())
```

```

assert(m.countOptions() == 15)
assert(m.countVariables() == 50)

% Add row -----
---
r = rowConventional('Feature 4');
r = r.addOption(option('Option 1', {'x1', 'x2', 'x3', 'x4'}));
r = r.addOption(option('Option 2', {'x1', 'x2', 'x3'}));
r = r.addOptionFromMorph('Option 3', m);
m = m.addRow(r); % Add row to matrix

% Add row -----
---
r = rowConventional('Feature 5');
r = r.addOption(option('Option 1', {'x1', 'x2', 'x3', 'x4'}));
r = r.addOption(option('Option 2', {'x1', 'x2', 'x3', 'x4'}));
r = r.addOption(option('Option 3', {'x1', 'x2', 'x3', 'x4'}));
m = m.addRow(r); % Add row to matrix

assert(size(m.getRows(), 1) == 5)

m = m.removeRow('Feature 5');
assert(size(m.getRows(), 1) == 4)
end

function testReduction(~)
% Initialize morphological matrix
m = morphologicalMatrix('Morphological Matrix');

% Add row -----
---
r = rowConventional('Feature 1');
r = r.addOption(option('Option 1', {'x1', 'x2', 'x3', 'x4'}));
r = r.addOption(option('Option 2', {'x1', 'x2', 'x3'}));
r = r.addOption(option('Option 3', {'x1', 'x2', 'x3'}));
r = r.addOption(option('Option 4', {'x5'}));
m = m.addRow(r); % Add row to matrix

% Add row -----
---
r = rowConventional('Feature 2');
r = r.addOption(option('Option 1', {'x1', 'x2', 'x3', 'x4'}));
r = r.addOption(option('Option 2', {'x1', 'x2', 'x3'}));
r = r.addOption(option('Option 3', {'x1', 'x2', 'x3'}));
m = m.addRow(r); % Add row to matrix

% Add row -----
---
r = rowConventional('Feature 3');
r = r.addOption(option('Option 1', {'x1', 'x2', 'x3', 'x4'}));
r = r.addOption(option('Option 2', {'x1', 'x2', 'x3', 'x4'}));
r = r.addOption(option('Option 3', {'x1', 'x2', 'x3', 'x4'}));
m = m.addRow(r); % Add row to matrix

```



```

% Morphological reduction -----
---
calls1 = m.computeAlternatives;
assert(calls1 == 36)
fprintf('Before: %d\n', calls1)

[m, ~, removedVars] = m.reduce;

calls2 = m.computeAlternatives;
assert(calls2 == 6)
fprintf('After:  %d (%.2f%%)\n', calls2, 100*(calls1-calls2)/calls1)
fprintf('Removed %d optimizer calls\n',calls1-calls2)
fprintf('Added %d variables to the optimizer\n',removedVars)

assert(size(m.getRows(),1) == 2)
assert(calls2 < calls1)
end

function testCompatibility(~)
% Initialize morphological matrix
m = morphologicalMatrix('Morphological Matrix');

% Add row -----
---
r = rowConventional('Feature 1');
r = r.addOption(option('Option 1',{}));
r = r.addOption(option('Option 2',{}));
r = r.addOption(option('Option 3',{}));
m = m.addRow(r); % Add row to matrix

% Add row -----
---
r = rowConventional('Feature 2');
r = r.addOption(option('Option 1',{}));
r = r.addOption(option('Option 2',{}));
r = r.addOption(option('Option 3',{}));
m = m.addRow(r); % Add row to matrix

% Add row -----
---
r = rowConventional('Feature 3');
r = r.addOption(option('Option 1',{}));
r = r.addOption(option('Option 2',{}));
r = r.addOption(option('Option 3',{}));
m = m.addRow(r); % Add row to matrix

%% Compatibility matrix
m = m.setUpCompatibilityIndices;
c = m.initializeCompatibilityMatrix;

c(1,4) = 0;
c(1,5) = 0;

c(1,7) = 0;
c(1,8) = 0;

```

```

% Symmetrize matrix
c = symmetrize(c);

assert(m.computeAlternatives == 27)
assert(m.computeCompatibleAlternatives(c,[],0) == 19)
end

%% Optional file fixtures
function setUpOnce(~) % do not change function name
% Prepare workspace
close all
clear global
clear

addpath('../morphologicalTree')
end

function tearDownOnce(~) % do not change function name
% change back to original path, for example
end

%% Optional fresh fixtures
function setUp(~) % do not change function name
end

function tearDown(~) % do not change function name
end

```

B.3.3 Example scripts

A few examples demonstrating how to use the data structures presented here above.

exampleTree.m

```

% Prepare workspace
clc
close all
clear

addpath('../morphologicalTree')
addpath('../morphologicalTree/interfaces')
addpath('../morphologicalTree/tinevez-matlab-tree-3d13d15')

airship = createMM_Macro();
fw = createMM_FixedWing();
macro = createMM_Macro();
multirotor = createMM_Multirotor();
ornithopter = createMM_Ornithopter();

%% Build morphological tree

```

```

t = morphologicalTree(createMM_Macro('Macroscopic level',false));

[ t, node1 ] = t.addnode(1, createMM_FixedWing('Fixed-Wing',true)); %
Abstract
[ t, node11 ] = t.addnode(node1, createMM_FixedWing('Vehicle
1',false));
[ t, node12 ] = t.addnode(node1, createMM_FixedWing('Vehicle
2',false));
[ t, node13 ] = t.addnode(node1, createMM_FixedWing('Vehicle
3',false));

[ t, node2 ] = t.addnode(1, createMM_Multirotor('Multirotor',true)); %
Abstract
[ t, node21 ] = t.addnode(node2, createMM_Multirotor('Vehicle
1',false));
[ t, node22 ] = t.addnode(node2, createMM_Multirotor('Vehicle
2',false));
[ t, node23 ] = t.addnode(node2, createMM_Multirotor('Vehicle
3',false));

[ t, node3 ] = t.addnode(1, createMM_Airship('Airship',true)); %
Abstract
[ t, node31 ] = t.addnode(node3, createMM_Airship('Vehicle 1',false));

[ t, node4 ] = t.addnode(1, createMM_Ornithopter('Ornithopter',true));
% Abstract
[ t, node41 ] = t.addnode(node4, createMM_Ornithopter('Vehicle
1',false));

t.tostring

%% Operations
before = t.computeAlternatives;
[t,bool,nVars,nOptions] = t.reduce;
after = t.computeAlternatives;
fprintf('%g/%g (%%f%%)\n',after,before,100*(before - after)/before)
fprintf('%d variables transferred to the optimizer\n',nVars)
fprintf('%d options removed\n',nOptions)
fprintf('\n')

```

exampleCompatibility.m

```

% Prepare workspace
clc
close all
clear

addpath('../morphologicalTree')

%% Macroscopic level
m = morphologicalMatrix('Macroscopic level');

```

```

r = rowConventional('Mission type');
r = r.addOption(option('HALE', {}));
r = r.addOption(option('Long-range strike', {}));
r = r.addOption(option('MALE', {}));
r = r.addOption(option('Close-range support', {}));
r = r.addOption(option('MUAV', {}));
r = r.addOption(option('MAV', {}));
m = m.addRow(r);

r = rowConventional('Architecture');
r = r.addOption(option('Fixed/Conventional', {}));
r = r.addOption(option('Product family', {}));
r = r.addOption(option('Scale-based product family', {}));
r = r.addOption(option('Reconfigurable', {}));
r = r.addOption(option('Online reconfigurable', {}));
r = r.addOption(option('Modular', {}));
m = m.addRow(r);

r = rowConventional('Control type');
r = r.addOption(option('Centralized', {}));
r = r.addOption(option('Decentralized', {}));
r = r.addOption(option('Hybrid', {}));
m = m.addRow(r);

r = rowConventional('Control scheme');
r = r.addOption(option('Leader/Follower', {}));
r = r.addOption(option('Consensus', {}));
r = r.addOption(option('Partitioned', {}));
r = r.addOption(option('Distributed', {}));
r = r.addOption(option('Hierarchical', {}));
m = m.addRow(r);

r = rowConventional('Ground station');
r = r.addOption(option('Remote base', {}));
r = r.addOption(option('Laptop', {}));
r = r.addOption(option('Wearable technology', {}));
m = m.addRow(r);

m = m.setUpCompatibilityIndices;
fprintf(m.toString)

%% Compatibility matrix
n = m.countOptions;

% Assume all compatible
c = m.initializeCompatibilityMatrix;
sum(sum(c == 0))

% Fill in incompatibilities
c(13,17) = 0; c(13,18) = 0; c(13,19) = 0; c(13,20) = 0;
c(14,16) = 0; c(14,20) = 0;
c(15,16) = 0; c(15,17) = 0; c(15,18) = 0; c(15,19) = 0;

sum(sum(c == 0))

```

```

%% Symmetrize matrix
c = symmetrize(c);

% Total alternatives
m.computeAlternatives

figure
pcolor(c)
colormap(gray(2))
axis ij
axis square
set(gca, 'FontName', 'Times New Roman', 'FontSize', 18)

% Compatible alternatives
tic
nn = m.computeCompatibleAlternatives(c, [], 0)
toc

```

exampleReduction.m

```

% Prepare workspace
clc
close all
clear

addpath('../morphologicalTree')
addpath('../morphologicalTree/interfaces')

%% Analysis
fprintf('Macro =====\n')
a = createMM_Macro('Macroscopic level', false);
before = a.computeAlternatives;
[a,~,nVars,nOptions] = a.reduce;
after = a.computeAlternatives;
fprintf('%g/%g (%f%%)\n', after, before, 100*(before - after)/before)
fprintf('%d variables transferred to the optimizer\n', nVars)
fprintf('%d options removed\n', nOptions)
fprintf('\n')

fprintf('Fixed-Wing =====\n')
a = createMM_FixedWing('Fixed-Wing', false);
before = a.computeAlternatives;
a.countOptions
a.countVariables
[a,~,nVars,nOptions] = a.reduce;
after = a.computeAlternatives;
fprintf('%g/%g (%f%%)\n', after, before, 100*(before - after)/before)
fprintf('%d variables transferred to the optimizer\n', nVars)
fprintf('%d options removed\n', nOptions)
fprintf('\n')

fprintf('Multirotor =====\n')
a = createMM_Multirotor('Multirotor', false);

```

```

before = a.computeAlternatives;
a.countOptions
a.countVariables
[a,~,nVars,nOptions] = a.reduce;
after = a.computeAlternatives;
fprintf('%g/%g (%f%%)\n',after,before,100*(before - after)/before)
fprintf('%d variables transferred to the optimizer\n',nVars)
fprintf('%d options removed\n',nOptions)
fprintf('\n')

fprintf('Airship =====\n')
a = createMM_Airship('Airship',false);
before = a.computeAlternatives;
a.countOptions
a.countVariables
[a,~,nVars,nOptions] = a.reduce;
after = a.computeAlternatives;
fprintf('%g/%g (%f%%)\n',after,before,100*(before - after)/before)
fprintf('%d variables transferred to the optimizer\n',nVars)
fprintf('%d options removed\n',nOptions)
fprintf('\n')

fprintf('Ornithopter =====\n')
a = createMM_Ornithopter('Ornithopter',false);
before = a.computeAlternatives;
a.countOptions
a.countVariables
[a,~,nVars,nOptions] = a.reduce;
after = a.computeAlternatives;
fprintf('%g/%g (%f%%)\n',after,before,100*(before - after)/before)
fprintf('%d variables transferred to the optimizer\n',nVars)
fprintf('%d options removed\n',nOptions)
fprintf('\n')

```

B.3.4 Interfaces

A set of functions creating notional morphological matrices and compatibility matrices for a number of aircraft architectures and for a notional group composition.

createMM_Airship.m

```

function [m,c] = createMM_Airship(name,abstract)
if nargin < 2
    abstract = false;
    name = 'Airship';
end

% Morphological matrix

```

```

m = morphologicalMatrix(name, abstract);

r = rowConventional('Lifting medium');
r = r.addOption(option('Cold gas (Helium)', {}));
r = r.addOption(option('Hot air', {}));
m = m.addRow(r);

r = rowConventional('Empennage configuration');
r = r.addOption(option('Y', {'nTails', 'deltaAngle', 'offsetAngle'}));
r = r.addOption(option('Inverted
Y', {'nTails', 'deltaAngle', 'offsetAngle'}));
r = r.addOption(option('X', {'nTails', 'deltaAngle', 'offsetAngle'}));
r = r.addOption(option('Cross', {'nTails', 'deltaAngle', 'offsetAngle'}));
m = m.addRow(r);

r = rowConventional('Ballonet-based pitch trim');
r = r.addOption(option('Yes', {'nBallonets'}));
r = r.addOption(option('No', {'nBallonets'}));
m = m.addRow(r);

r = rowConventional('Energy source');
r = r.addOption(option('Bio-chemical', {}));
r = r.addOption(option('Electric charge', {}));
r = r.addOption(option('Electrolyte', {}));
r = r.addOption(option('Hybrid', {}));
m = m.addRow(r);

r = rowConventional('Energy storage');
% Battery variables include
% Battery conditions: 'C-rate', 'E-
rate', 'SOC', 'DOD', 'terminalV', 'opencircuitV', 'R'
% Technical specs: nominal voltage, cut-off voltage, capacity C,
energy,
% cycle life, specific energy, specific power, energy density, power
% density, maximum continuous discharge current, maximum 30s discharge
% pulse current, charge voltage, float voltage, charge current
% Source: http://web.mit.edu/evt/summary\_battery\_specifications.pdf
r = r.addOption(option('Battery (LiCoO2)', {'batteryVariables'}));
r = r.addOption(option('Battery (LiFePO4)', {'batteryVariables'}));
r = r.addOption(option('Battery (LiPo)', {'batteryVariables'}));
r = r.addOption(option('Battery (NiCad)', {'batteryVariables'}));
r = r.addOption(option('Battery (NiMH)', {'batteryVariables'}));
r = r.addOption(option('Fuel tank', {'volume'}));
r = r.addOption(option('External fuel tank', {'size', 'position'}));
r = r.addOption(option('Electrolyte tank', {'volume'}));
r = r.addOption(option('Fuel tank + Battery
(LiCoO2)', {'volume', 'batteryVariables'}));
r = r.addOption(option('Fuel tank + Battery
(LiFePO4)', {'volume', 'batteryVariables'}));
r = r.addOption(option('Fuel tank + Battery
(LiPo)', {'volume', 'batteryVariables'}));
r = r.addOption(option('Fuel tank + Battery
(NiCad)', {'volume', 'batteryVariables'}));
r = r.addOption(option('Fuel tank + Battery
(NiMH)', {'volume', 'batteryVariables'}));

```

```

r = r.addOption(option('External fuel tank + Battery
(LiCoO2)', {'size', 'position', 'batteryVariables'}));
r = r.addOption(option('External fuel tank + Battery
(LiFePO4)', {'size', 'position', 'batteryVariables'}));
r = r.addOption(option('External fuel tank + Battery
(LiPo)', {'size', 'position', 'batteryVariables'}));
r = r.addOption(option('External fuel tank + Battery
(NiCad)', {'size', 'position', 'batteryVariables'}));
r = r.addOption(option('External fuel tank + Battery
(NiMH)', {'size', 'position', 'batteryVariables'}));
m = m.addRow(r);

r = rowConventional('Converter to mechanical energy');
r = r.addOption(option('Piston', {}));
r = r.addOption(option('Turbine', {}));
r = r.addOption(option('Electric motor (Spinning cage, Brushed)', {}));
r = r.addOption(option('Electric motor (Spinning cage,
Brushless)', {}));
r = r.addOption(option('Electric motor (Spinning shaft, Brushed)', {}));
r = r.addOption(option('Electric motor (Spinning shaft,
Brushless)', {}));
r = r.addOption(option('Hybrid (Piston/Electric)', {}));
r = r.addOption(option('Hybrid (Turbine/Electric)', {}));
r = r.addOption(option('Fuel cell and electric motor', {}));
m = m.addRow(r);

r = rowConventional('Number of rotors');
r = r.addOption(option('1', {'nRotors', 'positionRotors'}));
r = r.addOption(option('2', {'nRotors', 'positionRotors'}));
r = r.addOption(option('3', {'nRotors', 'positionRotors'}));
r = r.addOption(option('4', {'nRotors', 'positionRotors'}));
r = r.addOption(option('6', {'nRotors', 'positionRotors'}));
r = r.addOption(option('8', {'nRotors', 'positionRotors'}));
m = m.addRow(r);

r = rowConventional('Steerable propulsion');
r = r.addOption(option('Yes', {}));
r = r.addOption(option('No', {}));
m = m.addRow(r);

r = rowConventional('Converter to lift/thrust');
r = r.addOption(option('Rotor', {}));
r = r.addOption(option('Fan', {}));
r = r.addOption(option('Propeller', {}));
m = m.addRow(r);

r = rowConventional('Number of blades');
r = r.addOption(option('2', {'nBlades'}));
r = r.addOption(option('3', {'nBlades'}));
r = r.addOption(option('4', {'nBlades'}));
r = r.addOption(option('6', {'nBlades'}));
r = r.addOption(option('8', {'nBlades'}));
r = r.addOption(option('10', {'nBlades'}));
m = m.addRow(r);

r = rowConventional('Blade type');

```



```

r = r.addOption(option('Fixed pitch', {}));
r = r.addOption(option('Variable pitch', {}));
m = m.addRow(r);

r = rowCombinatorial('Imaging');
r = r.addOption(option('Mono (Fixed-mount)', {}));
r = r.addOption(option('Mono (Gyro-stabilized)', {}));
r = r.addOption(option('RGB camera (Fixed-mount)', {}));
r = r.addOption(option('RGB camera (Gyro-stabilized)', {}));
r = r.addOption(option('Multispectral data (Fixed-mount)', {}));
r = r.addOption(option('Multispectral data (Gyro-stabilized)', {}));
r = r.addOption(option('Thermal camera (Fixed-mount)', {}));
r = r.addOption(option('Thermal camera (Gyro-stabilized)', {}));
m = m.addRow(r);

r = rowCombinatorial('Mapping');
r = r.addOption(option('2D LIDAR', {}));
r = r.addOption(option('3D LIDAR', {}));
r = r.addOption(option('Sonar', {}));
m = m.addRow(r);

r = rowCombinatorial('Attitude');
r = r.addOption(option('IMU+GPS', {}));
m = m.addRow(r);

r = rowCombinatorial('Altitude');
r = r.addOption(option('GPS', {}));
r = r.addOption(option('Barometer', {}));
r = r.addOption(option('Sonar', {}));
m = m.addRow(r);

r = rowConventional('Communications');
r = r.addOption(option('Radio', {}));
m = m.addRow(r);

r = rowConventional('Hull type');
r = r.addOption(option('Non-rigid', {}));
r = r.addOption(option('Semi-rigid', {}));
r = r.addOption(option('Rigid', {}));
m = m.addRow(r);

r = rowConventional('Battens');
r = r.addOption(option('Yes', {}));
r = r.addOption(option('No', {}));
m = m.addRow(r);

% Compatibility matrix
=====

% Initialization
m = m.setUpCompatibilityIndices;
c = m.initializeCompatibilityMatrix;

% Fill in incompatibilities

```

```
% Energy sources
```

```
---
```

```
c(9,13) = 0;  
c(9,14) = 0;  
c(9,15) = 0;  
c(9,16) = 0;  
c(9,17) = 0;  
c(9,20) = 0;  
c(9,21) = 0;  
c(9,22) = 0;  
c(9,23) = 0;  
c(9,24) = 0;  
c(9,25) = 0;  
c(9,26) = 0;  
c(9,27) = 0;  
c(9,28) = 0;  
c(9,29) = 0;  
c(9,30) = 0;
```

```
c(9,33) = 0;  
c(9,34) = 0;  
c(9,35) = 0;  
c(9,36) = 0;  
c(9,37) = 0;  
c(9,38) = 0;  
c(9,39) = 0;
```

```
c(10,18) = 0;  
c(10,19) = 0;  
c(10,20) = 0;  
c(10,21) = 0;  
c(10,22) = 0;  
c(10,23) = 0;  
c(10,24) = 0;  
c(10,25) = 0;  
c(10,26) = 0;  
c(10,27) = 0;  
c(10,28) = 0;  
c(10,29) = 0;  
c(10,30) = 0;
```

```
c(10,31) = 0;  
c(10,32) = 0;  
c(10,37) = 0;  
c(10,38) = 0;  
c(10,39) = 0;
```

```
c(11,13) = 0;  
c(11,14) = 0;  
c(11,15) = 0;  
c(11,16) = 0;  
c(11,17) = 0;  
c(11,18) = 0;  
c(11,19) = 0;  
c(11,21) = 0;  
c(11,22) = 0;
```

```

c(11,23) = 0;
c(11,24) = 0;
c(11,25) = 0;
c(11,26) = 0;
c(11,27) = 0;
c(11,28) = 0;
c(11,29) = 0;
c(11,30) = 0;

c(11,31) = 0;
c(11,32) = 0;
c(11,33) = 0;
c(11,34) = 0;
c(11,35) = 0;
c(11,36) = 0;
c(11,37) = 0;
c(11,38) = 0;

c(12,13) = 0;
c(12,14) = 0;
c(12,15) = 0;
c(12,16) = 0;
c(12,17) = 0;
c(12,18) = 0;
c(12,19) = 0;
c(12,20) = 0;

c(12,31) = 0;
c(12,32) = 0;
c(12,33) = 0;
c(12,34) = 0;
c(12,35) = 0;
c(12,36) = 0;
c(12,39) = 0;

% Energy storage -----
---
c(13,31) = 0; c(13,32) = 0; c(13,37) = 0; c(13,38) = 0; c(13,39) = 0;
c(14,31) = 0; c(14,32) = 0; c(14,37) = 0; c(14,38) = 0; c(14,39) = 0;
c(15,31) = 0; c(15,32) = 0; c(15,37) = 0; c(15,38) = 0; c(15,39) = 0;
c(16,31) = 0; c(16,32) = 0; c(16,37) = 0; c(16,38) = 0; c(16,39) = 0;
c(17,31) = 0; c(17,32) = 0; c(17,37) = 0; c(17,38) = 0; c(17,39) = 0;

c(18,33) = 0; c(18,34) = 0; c(18,35) = 0; c(18,36) = 0; c(18,37) = 0;
c(18,38) = 0; c(18,39) = 0;
c(19,33) = 0; c(19,34) = 0; c(19,35) = 0; c(19,36) = 0; c(19,37) = 0;
c(19,38) = 0; c(19,39) = 0;

c(20,31) = 0;
c(20,32) = 0;
c(20,33) = 0;
c(20,34) = 0;
c(20,35) = 0;
c(20,36) = 0;
c(20,37) = 0;
c(20,38) = 0;

```

```

for i = 21:30
    c(i,31) = 0;
    c(i,32) = 0;
    c(i,33) = 0;
    c(i,34) = 0;
    c(i,35) = 0;
    c(i,36) = 0;
    c(i,39) = 0;
end

```

```

% Hull type -----
---
c(75,78) = 0;

% Symmetrize matrix
=====
c = m.symmetrize(c);
end

```

createMM_FixedWing.m

```

function [m,c] = createMM_FixedWing(name,abstract)
if nargin < 2
    abstract = false;
    name = 'Fixed-Wing';
end

% Empennage
mEmpennage = morphologicalMatrix('Empennage');

r = rowConventional('Number of tailplanes');
r = r.addOption(option('0 (Tailless)', {'variablesTailless'}));
r = r.addOption(option('0 (Canard)', {'VariablesCanard'}));
r = r.addOption(option('1', {'nTailPlanes'}));
r = r.addOption(option('2', {'nTailPlanes'}));
r = r.addOption(option('3', {'nTailPlanes'}));
mEmpennage = mEmpennage.addRow(r); % Add row to matrix

r = rowConventional('Location');
r = r.addOption(option('Low', {'variablesLow'}));
r = r.addOption(option('Mid', {'variablesMid'}));
r = r.addOption(option('High', {'variablesHigh'}));
r = r.addOption(option('Booms (Tail)', {'variablesBoomsTail'}));
r = r.addOption(option('Booms (Wing)', {'variablesBoomsWing'}));
mEmpennage = mEmpennage.addRow(r); % Add row to matrix

r = rowConventional('Moving surfaces');
r = r.addOption(option('Independent', {'variablesIndependent'}));
r = r.addOption(option('Stabilator', {'variablesStabilator'}));
mEmpennage = mEmpennage.addRow(r); % Add row to matrix

r = rowConventional('Number of fins');

```

```

r = r.addOption(option('0',{ 'nFins','positions' }));
r = r.addOption(option('1',{ 'nFins','positions' }));
r = r.addOption(option('2',{ 'nFins','positions' }));
mEmpennage = mEmpennage.addRow(r); % Add row to matrix

r = rowConventional('Configuration');
r = r.addOption(option('Fin/Taiplane',{ 'nTailPlanes','positions' }));
r = r.addOption(option('V tail',{ 'nTailPlanes','positions' }));
r = r.addOption(option('Inverted V tail',{ 'nTailPlanes','positions' }));
r = r.addOption(option('X tail',{ 'nTailPlanes','positions' }));
r = r.addOption(option('Pelikan',{ 'variablesPelikan' }));
mEmpennage = mEmpennage.addRow(r); % Add row to matrix

% Fixed wing
=====
m = morphologicalMatrix(name,abstract);

r = rowConventional('Body');
r = r.addOption(option('Separate fuselage',{ }));
r = r.addOption(option('Flying wing',{ }));
r = r.addOption(option('Blended body',{ }));
r = r.addOption(option('Lifting body',{ }));
m = m.addRow(r);

r = rowConventional('Wing');
% Wing variables include
% Aspect ratio, span, sweep angle 1, 2, taper ratio, chord length
r = r.addOption(option('Straight',{ 'wingVariables' }));
r = r.addOption(option('Swept',{ 'wingVariables' }));
r = r.addOption(option('Delta',{ 'wingVariables' }));
r = r.addOption(option('Compound delta',{ 'wingVariables' }));
m = m.addRow(r);

r = rowConventional('Wing position');
r = r.addOption(option('Low',{ }));
r = r.addOption(option('Mid',{ }));
r = r.addOption(option('Shoulder',{ }));
r = r.addOption(option('High',{ }));
r = r.addOption(option('Parasol',{ }));
m = m.addRow(r);

r = rowConventional('Detachable wing');
r = r.addOption(option('Yes',{ 'weightDetachableMechanism' }));
r = r.addOption(option('No',{ 'weightDetachableMechanism' }));
m = m.addRow(r);

r = rowConventional('Empennage');
r = r.addOption(option('None',{ }));
r = r.addOptionFromMorph('Empennage',mEmpennage);
m = m.addRow(r);

r = rowConventional('Type of launch');
r = r.addOption(option('Horizontal',{ 'vInit','pitchInit','hInit' }));
r = r.addOption(option('Vertical',{ 'vInit','pitchInit','hInit' }));
r = r.addOption(option('Hand-launched',{ 'vInit','pitchInit','hInit' }));

```

```

r = r.addOption(option('Aircraft-
launched',{ 'vInit', 'pitchInit', 'hInit'}));
r = r.addOption(option('Catapult-
launched',{ 'vInit', 'pitchInit', 'hInit'}));
m = m.addRow(r);

r = rowConventional('Type of landing');
r = r.addOption(option('Horizontal landing',{ }));
r = r.addOption(option('Vertical landing',{ }));
r = r.addOption(option('Energy dissipation crash',{ }));
r = r.addOption(option('Parachute',{ }));
r = r.addOption(option('Net',{ }));
m = m.addRow(r);

r = rowConventional('Number of motors');
r = r.addOption(option('1',{ }));
r = r.addOption(option('2',{ 'nEnginePairs'}));
r = r.addOption(option('3',{ }));
r = r.addOption(option('4',{ 'nEnginePairs'}));
m = m.addRow(r);

r = rowConventional('Energy source');
r = r.addOption(option('Bio-chemical',{ }));
r = r.addOption(option('Electric charge',{ }));
r = r.addOption(option('Solar',{ }));
r = r.addOption(option('Electrolyte',{ }));
r = r.addOption(option('Hybrid',{ }));
m = m.addRow(r);

% Battery variables include
% Battery conditions: 'C-rate','E-
rate','SOC','DOD','terminalV','opencircuitV','R'
% Technical specs: nominal voltage, cut-off voltage, capacity C,
energy,
% cycle life, specific energy, specific power, energy density, power
% density, maximum continuous discharge current, maximum 30s discharge
% pulse current, charge voltage, float voltage, charge current
% Source: http://web.mit.edu/evt/summary\_battery\_specifications.pdf
r = rowConventional('Energy storage');
r = r.addOption(option('Battery (LiCo2)',{ 'batteryVariables'}));
r = r.addOption(option('Battery (LiFePO4)',{ 'batteryVariables'}));
r = r.addOption(option('Battery (LiPo)',{ 'batteryVariables'}));
r = r.addOption(option('Battery (NiCad)',{ 'batteryVariables'}));
r = r.addOption(option('Battery (NiMH)',{ 'batteryVariables'}));
r = r.addOption(option('Fuel tank',{ 'volume'}));
r = r.addOption(option('External fuel tank',{ 'size','position'}));
r = r.addOption(option('Electrolyte tank',{ 'volume'}));
r = r.addOption(option('Fuel tank + Battery
(LiCo2)',{ 'volume','batteryVariables'}));
r = r.addOption(option('Fuel tank + Battery
(LiFePO4)',{ 'volume','batteryVariables'}));
r = r.addOption(option('Fuel tank + Battery
(LiPo)',{ 'volume','batteryVariables'}));
r = r.addOption(option('Fuel tank + Battery
(NiCad)',{ 'volume','batteryVariables'}));

```

```

r = r.addOption(option('Fuel tank + Battery
(NiMH)', {'volume', 'batteryVariables'}));
r = r.addOption(option('External fuel tank + Battery
(LiCoO2)', {'size', 'position', 'batteryVariables'}));
r = r.addOption(option('External fuel tank + Battery
(LiFePO4)', {'size', 'position', 'batteryVariables'}));
r = r.addOption(option('External fuel tank + Battery
(LiPo)', {'size', 'position', 'batteryVariables'}));
r = r.addOption(option('External fuel tank + Battery
(NiCad)', {'size', 'position', 'batteryVariables'}));
r = r.addOption(option('External fuel tank + Battery
(NiMH)', {'size', 'position', 'batteryVariables'}));
m = m.addRow(r);

r = rowConventional('Converter to mechanical energy');
r = r.addOption(option('Piston', {}));
r = r.addOption(option('Turbine', {}));
r = r.addOption(option('Electric motor', {}));
r = r.addOption(option('Hybrid (Piston/Electric)', {}));
r = r.addOption(option('Hybrid (Turbine/Electric)', {}));
r = r.addOption(option('Fuel cell and electric motor', {}));
m = m.addRow(r);

r = rowConventional('Converter to lift/thrust');
r = r.addOption(option('Rotor', {}));
r = r.addOption(option('Fan', {}));
r = r.addOption(option('Propeller', {}));
r = r.addOption(option('Jet', {}));
m = m.addRow(r);

r = rowCombinatorial('Imaging');
r = r.addOption(option('Mono (Fixed-mount)', {}));
r = r.addOption(option('Mono (Gyro-stabilized)', {}));
r = r.addOption(option('RGB camera (Fixed-mount)', {}));
r = r.addOption(option('RGB camera (Gyro-stabilized)', {}));
r = r.addOption(option('Multispectral data (Fixed-mount)', {}));
r = r.addOption(option('Multispectral data (Gyro-stabilized)', {}));
r = r.addOption(option('Thermal camera (Fixed-mount)', {}));
r = r.addOption(option('Thermal camera (Gyro-stabilized)', {}));
m = m.addRow(r);

r = rowCombinatorial('Mapping');
r = r.addOption(option('2D LIDAR', {}));
r = r.addOption(option('3D LIDAR', {}));
r = r.addOption(option('Sonar', {}));
m = m.addRow(r);

r = rowConventional('Attitude');
r = r.addOption(option('IMU+GPS', {}));
m = m.addRow(r);

r = rowCombinatorial('Altitude');
r = r.addOption(option('GPS', {}));
r = r.addOption(option('Barometer', {}));
r = r.addOption(option('Sonar', {}));
m = m.addRow(r);

```

```

r = rowConventional('Communications');
r = r.addOption(option('Radio', {}));
m = m.addRow(r);

r = rowConventional('Landing gear arrangement');
r = r.addOption(option('None', {}));
r = r.addOption(option('Wheels (Tail wheel)', {}));
r = r.addOption(option('Wheels (Tandem)', {}));
r = r.addOption(option('Wheels (Tricycle)', {}));
r = r.addOption(option('Wheels (Wing)', {}));
r = r.addOption(option('Skids', {}));
r = r.addOption(option('Floaters', {}));
r = r.addOption(option('Skis', {}));
m = m.addRow(r);

r = rowConventional('Landing gear type');
r = r.addOption(option('None', {}));
r = r.addOption(option('Fixed', {}));
r = r.addOption(option('Retractable', {}));
m = m.addRow(r);

r = rowConventional('Landing gear shock absorption');
r = r.addOption(option('None', {}));
r = r.addOption(option('Rigid', {}));
r = r.addOption(option('Leaf-type', {}));
r = r.addOption(option('Bungee cord', {}));
r = r.addOption(option('Shock struts', {}));
m = m.addRow(r);

% Compatibility matrix
=====

% Initialization
m = m.setUpCompatibilityIndices;
c = m.initializeCompatibilityMatrix;

% Fill in incompatibilities
c(2:4, [9, 11:13]) = 0;
c(2:4, 17:766) = 0;

c(774, 831:837) = 0; c(774, 839:840) = 0; c(774, 842:845) = 0;
c(776, 831:837) = 0; c(776, 842:845) = 0;

c(781, 786:790) = 0; c(781, 793:803) = 0;
c(781, 806:809) = 0;

c(782, 791:803) = 0;
c(782:783, 804:805) = 0; c(782:783, 807:809) = 0;

c(784, 786:792) = 0; c(784, 794:803) = 0;
c(784, 804:808) = 0;

c(785, 786:793) = 0;

```



```

c(785,804:806) = 0; c(785,809) = 0;

c(786:790,804:805) = 0; c(786:790,807:809) = 0;

c(791:792,806:809) = 0;

c(793,804:808) = 0;

c(794:803,804:806) = 0; c(794:803,809) = 0;

% Symmetrize matrix
=====
c = m.symmetrize(c);
end

```

createMM_Macro.m

```

function [m,c] = createMM_Macro(name,abstract)
% Macroscopic level
if nargin < 2
    abstract = false;
    name = 'Macroscopic level';
end
m = morphologicalMatrix(name,abstract);

r = rowConventional('Mission type');
r = r.addOption(option('HALE',{ }));
r = r.addOption(option('Long-range strike',{ }));
r = r.addOption(option('MALE',{ }));
r = r.addOption(option('Close-range support',{ }));
r = r.addOption(option('MUAV',{ }));
r = r.addOption(option('MAV',{ }));
m = m.addRow(r);

r = rowConventional('Architecture');
r = r.addOption(option('Fixed/Conventional',{ }));
r = r.addOption(option('Product family',{ }));
r = r.addOption(option('Scale-based product family',{ }));
r = r.addOption(option('Reconfigurable',{ }));
r = r.addOption(option('Online reconfigurable',{ }));
r = r.addOption(option('Modular',{ }));
m = m.addRow(r);

r = rowConventional('Control type');
r = r.addOption(option('Centralized',{ }));
r = r.addOption(option('Decentralized',{ }));
r = r.addOption(option('Hybrid',{ }));
m = m.addRow(r);

r = rowConventional('Control scheme');
r = r.addOption(option('Leader/Follower',{ }));
r = r.addOption(option('Consensus',{ }));

```

```

r = r.addOption(option('Partitioned', {}));
r = r.addOption(option('Distributed', {}));
r = r.addOption(option('Hierarchical', {}));
m = m.addRow(r);

r = rowConventional('Ground station');
r = r.addOption(option('Remote base', {}));
r = r.addOption(option('Laptop', {}));
r = r.addOption(option('Wearable technology', {}));
m = m.addRow(r);

% Compatibility matrix
=====
% Initialization (all compatible)
m = m.setUpCompatibilityIndices;
c = m.initializeCompatibilityMatrix;

% Fill in incompatibilities
c(13,17) = 0; c(13,18) = 0; c(13,19) = 0; c(13,20) = 0;
c(14,16) = 0; c(14,20) = 0;
c(15,16) = 0; c(15,17) = 0; c(15,18) = 0; c(15,19) = 0;

% Symmetrize matrix
=====
c = m.symmetrize(c);
end

```

createMM_Multirotor.m

```

function [m,c] = createMM_Multirotor(name,abstract)
% Multirotor
if nargin < 2
    abstract = false;
    name = 'Multirotor';
end

m = morphologicalMatrix(name,abstract);

r = rowConventional('Fairings');
r = r.addOption(option('None', {}));
r = r.addOption(option('Electronics only', {}));
r = r.addOption(option('Full body', {}));
r = r.addOption(option('Full body and payload', {}));
m = m.addRow(r);

r = rowConventional('Type of landing');
r = r.addOption(option('Vertical landing', {}));
r = r.addOption(option('Energy dissipation crash', {}));
r = r.addOption(option('Parachute', {}));
r = r.addOption(option('Net', {}));
m = m.addRow(r);

r = rowConventional('Energy source');

```

```

r = r.addOption(option('Bio-chemical', {}));
r = r.addOption(option('Electric charge', {}));
r = r.addOption(option('Electrolyte', {}));
r = r.addOption(option('Hybrid', {}));
m = m.addRow(r);

% Battery variables include
% Battery conditions: 'C-rate', 'E-
rate', 'SOC', 'DOD', 'terminalV', 'opencircuitV', 'R'
% Technical specs: nominal voltage, cut-off voltage, capacity C,
energy,
% cycle life, specific energy, specific power, energy density, power
% density, maximum continuous discharge current, maximum 30s discharge
% pulse current, charge voltage, float voltage, charge current
% Source: http://web.mit.edu/evt/summary\_battery\_specifications.pdf
r = rowConventional('Energy storage');
r = r.addOption(option('Battery (LiCoO2)', {'batteryVariables'}));
r = r.addOption(option('Battery (LiFePO4)', {'batteryVariables'}));
r = r.addOption(option('Battery (LiPo)', {'batteryVariables'}));
r = r.addOption(option('Battery (NiCad)', {'batteryVariables'}));
r = r.addOption(option('Battery (NiMH)', {'batteryVariables'}));
r = r.addOption(option('Fuel tank', {'volume'}));
r = r.addOption(option('External fuel tank', {'size', 'position'}));
r = r.addOption(option('Electrolyte tank', {'volume'}));
r = r.addOption(option('Fuel tank + Battery
(LiCoO2)', {'volume', 'batteryVariables'}));
r = r.addOption(option('Fuel tank + Battery
(LiFePO4)', {'volume', 'batteryVariables'}));
r = r.addOption(option('Fuel tank + Battery
(LiPo)', {'volume', 'batteryVariables'}));
r = r.addOption(option('Fuel tank + Battery
(NiCad)', {'volume', 'batteryVariables'}));
r = r.addOption(option('Fuel tank + Battery
(NiMH)', {'volume', 'batteryVariables'}));
r = r.addOption(option('External fuel tank + Battery
(LiCoO2)', {'size', 'position', 'batteryVariables'}));
r = r.addOption(option('External fuel tank + Battery
(LiFePO4)', {'size', 'position', 'batteryVariables'}));
r = r.addOption(option('External fuel tank + Battery
(LiPo)', {'size', 'position', 'batteryVariables'}));
r = r.addOption(option('External fuel tank + Battery
(NiCad)', {'size', 'position', 'batteryVariables'}));
r = r.addOption(option('External fuel tank + Battery
(NiMH)', {'size', 'position', 'batteryVariables'}));
m = m.addRow(r);

r = rowConventional('Converter to mechanical energy');
r = r.addOption(option('Piston', {}));
r = r.addOption(option('Turbine', {}));
r = r.addOption(option('Electric motor (Spinning cage, Brushed)', {}));
r = r.addOption(option('Electric motor (Spinning cage,
Brushless)', {}));
r = r.addOption(option('Electric motor (Spinning shaft, Brushed)', {}));
r = r.addOption(option('Electric motor (Spinning shaft,
Brushless)', {}));
r = r.addOption(option('Hybrid (Piston/Electric)', {}));

```

```

r = r.addOption(option('Hybrid (Turbine/Electric)', {}));
r = r.addOption(option('Fuel cell and electric motor', {}));
m = m.addRow(r);

r = rowConventional('Rotorcraft');
r = r.addOption(option('Single rotor', {}));
r = r.addOption(option('Coaxial', {}));
r = r.addOption(option('Tail sitter', {}));
r = r.addOption(option('Tilt-rotor', {}));
r = r.addOption(option('Multirotor', {}));
m = m.addRow(r);

r = rowConventional('Number of rotors');
r = r.addOption(option('1', {'nRotors'}));
r = r.addOption(option('2', {'nRotors'}));
r = r.addOption(option('3', {'nRotors'}));
r = r.addOption(option('4', {'nRotors'}));
r = r.addOption(option('6', {'nRotors'}));
r = r.addOption(option('8', {'nRotors'}));
m = m.addRow(r);

r = rowConventional('Converter to lift/thrust');
r = r.addOption(option('Rotor', {}));
r = r.addOption(option('Fan', {}));
r = r.addOption(option('Propeller', {}));
m = m.addRow(r);

r = rowConventional('Number of blades');
r = r.addOption(option('2', {'nBlades'}));
r = r.addOption(option('3', {'nBlades'}));
r = r.addOption(option('4', {'nBlades'}));
r = r.addOption(option('6', {'nBlades'}));
r = r.addOption(option('8', {'nBlades'}));
r = r.addOption(option('10', {'nBlades'}));
m = m.addRow(r);

r = rowConventional('Rotors/Frame arrangement');
r = r.addOption(option('I', {}));
r = r.addOption(option('X', {'armsAngles'}));
r = r.addOption(option('Y', {'armsAngles'}));
r = r.addOption(option('V', {'armsAngles'}));
m = m.addRow(r);

r = rowConventional('Blade type');
r = r.addOption(option('Fixed pitch', {}));
r = r.addOption(option('Variable pitch', {}));
m = m.addRow(r);

r = rowCombinatorial('Imaging');
r = r.addOption(option('Mono (Fixed-mount)', {}));
r = r.addOption(option('Mono (Gyro-stabilized)', {}));
r = r.addOption(option('RGB camera (Fixed-mount)', {}));
r = r.addOption(option('RGB camera (Gyro-stabilized)', {}));
r = r.addOption(option('Multispectral data (Fixed-mount)', {}));
r = r.addOption(option('Multispectral data (Gyro-stabilized)', {}));

```

```

r = r.addOption(option('Thermal camera (Fixed-mount)', {}));
r = r.addOption(option('Thermal camera (Gyro-stabilized)', {}));
m = m.addRow(r);

r = rowCombinatorial('Mapping');
r = r.addOption(option('2D LIDAR', {}));
r = r.addOption(option('3D LIDAR', {}));
r = r.addOption(option('Sonar', {}));
m = m.addRow(r);

r = rowConventional('Attitude');
r = r.addOption(option('IMU+GPS', {}));
m = m.addRow(r);

r = rowCombinatorial('Altitude');
r = r.addOption(option('GPS', {}));
r = r.addOption(option('Barometer', {}));
r = r.addOption(option('Sonar', {}));
m = m.addRow(r);

r = rowConventional('Communications');
r = r.addOption(option('Radio', {}));
m = m.addRow(r);

r = rowConventional('Landing gear arrangement');
r = r.addOption(option('None', {}));
r = r.addOption(option('Foam pads', {}));
r = r.addOption(option('Skids', {}));
r = r.addOption(option('Floaters', {}));
m = m.addRow(r);

r = rowConventional('Landing gear type');
r = r.addOption(option('None', {}));
r = r.addOption(option('Fixed', {}));
r = r.addOption(option('Retractable', {}));
m = m.addRow(r);

r = rowConventional('Landing gear shock absorption');
r = r.addOption(option('None', {}));
r = r.addOption(option('Rigid', {}));
r = r.addOption(option('Leaf-type', {}));
r = r.addOption(option('Bungee cord', {}));
r = r.addOption(option('Shock struts', {}));
m = m.addRow(r);

r = rowConventional('Frame type');
r = r.addOption(option('Aerial cinematography', {}));
r = r.addOption(option('Sport', {}));
r = r.addOption(option('Sport FPV', {}));
r = r.addOption(option('Mini', {}));
r = r.addOption(option('Mini FPV', {}));
m = m.addRow(r);

% Compatibility matrix
=====

```

```

% Assume all compatible
m = m.setUpCompatibilityIndices;
c = m.initializeCompatibilityMatrix;

% Fill in incompatibilities
c(6,83) = 0; c(6,84) = 0; c(6,85) = 0; c(6,87) = 0; c(6,88) = 0;
c(6,90:93) = 0;
c(8,83) = 0; c(8,84) = 0; c(8,85) = 0; c(8,90:93) = 0;

c(9,13:17) = 0; c(9,20:30) = 0;
c(9,33:39) = 0;

c(10,18:30) = 0;
c(10,31:32) = 0; c(10,37:39) = 0;

c(11,13:19) = 0; c(11,21:30) = 0;
c(11,31:38) = 0;

c(12,13:20) = 0;
c(12,31:36) = 0; c(12,39) = 0;

c(13:17,31:32) = 0;
c(13:17,37:39) = 0;

c(18:19,33:39) = 0;

c(20,31:38) = 0;

c(21:30,31:36) = 0;
c(21:30,39) = 0;

% Symmetrize matrix
=====
c = m.symmetrize(c);
end

```

createMM_Ornithopter.m

```

function [m,c] = createMM_Ornithopter(name,abstract)
if nargin < 2
    name = 'Ornithopter';
    abstract = false;
end

% Ornithopter
m = morphologicalMatrix(name,abstract);

r = rowConventional('Wing twisting');
r = r.addOption(option('Yes (Spar rotation)',{ }));
r = r.addOption(option('Yes (Spar torsion)',{ }));
r = r.addOption(option('Yes (Servo-controlled)',{ }));

```

```

r = r.addOption(option('Yes (Auxiliary spar)', {}));
r = r.addOption(option('No', {}));
m = m.addRow(r);

r = rowConventional('Wing type');
r = r.addOption(option('Flying wing', {}));
r = r.addOption(option('Tandem wing', {}));
r = r.addOption(option('Thrust wing', {}));
r = r.addOption(option('Oscillating stretched wing', {}));
r = r.addOption(option('Rotating wing', {}));
m = m.addRow(r);

r = rowConventional('Energy source');
r = r.addOption(option('Bio-chemical', {}));
r = r.addOption(option('Electric charge', {}));
r = r.addOption(option('Other', {}));
m = m.addRow(r);

% Battery variables include
% Battery conditions: 'C-rate', 'E-
rate', 'SOC', 'DOD', 'terminalV', 'opencircuitV', 'R'
% Technical specs: nominal voltage, cut-off voltage, capacity C,
energy,
% cycle life, specific energy, specific power, energy density, power
% density, maximum continuous discharge current, maximum 30s discharge
% pulse current, charge voltage, float voltage, charge current
% Source: http://web.mit.edu/evt/summary\_battery\_specifications.pdf
r = rowConventional('Energy storage');
r = r.addOption(option('Battery (LiCoO2)', {'batteryVariables'}));
r = r.addOption(option('Battery (LiFePO4)', {'batteryVariables'}));
r = r.addOption(option('Battery (LiPo)', {'batteryVariables'}));
r = r.addOption(option('Battery (NiCad)', {'batteryVariables'}));
r = r.addOption(option('Battery (NiMH)', {'batteryVariables'}));
r = r.addOption(option('Fuel tank', {}));
r = r.addOption(option('Rubber', {}));
m = m.addRow(r);

r = rowConventional('Gearbox type');
r = r.addOption(option('Strut', {}));
r = r.addOption(option('Plate', {}));
m = m.addRow(r);

r = rowConventional('Gear type');
r = r.addOption(option('Cluster', {}));
r = r.addOption(option('Spur with pinion wire', {}));
m = m.addRow(r);

r = rowConventional('Flapping mechanism');
r = r.addOption(option('Staggered crank', {}));
r = r.addOption(option('Outboard wing hinge', {}));
r = r.addOption(option('Dual cranks', {}));
r = r.addOption(option('Transverse shaft', {}));
m = m.addRow(r);

r = rowConventional('Converter to mechanical energy');

```

```

r = r.addOption(option('Internal combustion engine',{ }));
r = r.addOption(option('Electric motor (Spinning cage, Brushed)',{ }));
r = r.addOption(option('Electric motor (Spinning cage,
Brushless)',{ }));
r = r.addOption(option('Electric motor (Spinning shaft, Brushed)',{ }));
r = r.addOption(option('Electric motor (Spinning shaft,
Brushless)',{ }));
r = r.addOption(option('Rubber + shaft',{ }));
m = m.addRow(r);

r = rowCombinatorial('Imaging');
r = r.addOption(option('Mono',{ }));
r = r.addOption(option('RGB camera',{ }));
r = r.addOption(option('Multispectral data',{ }));
m = m.addRow(r);

r = rowCombinatorial('Attitude');
r = r.addOption(option('IMU+GPS',{ }));
m = m.addRow(r);

r = rowCombinatorial('Altitude');
r = r.addOption(option('GPS',{ }));
r = r.addOption(option('Barometer',{ }));
r = r.addOption(option('Sonar',{ }));
m = m.addRow(r);

r = rowConventional('Communications');
r = r.addOption(option('Radio',{ }));
m = m.addRow(r);

r = rowCombinatorial('Wing reinforcement');
r = r.addOption(option('Battens',{ }));
r = r.addOption(option('Perimeter',{ }));
m = m.addRow(r);

% Compatibility matrix
=====

% Assume all compatible
m = m.setUpCompatibilityIndices;
c = m.initializeCompatibilityMatrix;

% Fill in incompatibilities
c(11,14) = 0; c(11,15) = 0; c(11,16) = 0; c(11,17) = 0; c(11,18) = 0;
c(11,20) = 0;
c(11,30) = 0; c(11,31) = 0; c(11,32) = 0; c(11,33) = 0; c(11,34) = 0;

c(12,19) = 0; c(12,20) = 0;
c(12,29) = 0; c(12,34) = 0;

c(13,14) = 0; c(13,15) = 0; c(13,16) = 0; c(13,17) = 0; c(13,18) = 0;
c(13,19) = 0;
c(13,29) = 0; c(13,30) = 0; c(13,31) = 0; c(13,32) = 0; c(13,33) = 0;

c(14:18,29) = 0;

```



```

c(14:18,34) = 0;

c(19,30:34) = 0;
c(20,29:33) = 0;

% Symmetrize matrix
=====
c = m.symmetrize(c);
end

```

B.3.5 PACE cluster scripts

Due to the very demanding resources of the recursive compatibility function, some evaluations had to be performed on the Georgia Tech PACE cluster. This subsection gives examples of scripts that were produced to run such function calls on the clusters.

airship.pbs

The script submitted to the PACE cluster with the command *qsub*, it runs the script *airship.m* on the cluster.

```

#PBS -N airship4
#PBS -q enterprise
#PBS -o airship4.output.$PBS_JOBID
#PBS -j oe
#PBS -l nodes=10:ppn=24
#PBS -l mem=8gb
#PBS -l walltime=5:00:00
#PBS -m abe
#PBS -M jdurand7@gatech.edu

# Load Matlab
module load matlab/r2016a

# Change to working directory
cd scripts/airship

# Run code
matlab -nodisplay -nosplash -nodesktop -r "run airship.m"

```

airship.m

The actual Matlab script run on the cluster. It computes the number of compatible alternatives out of the morphological and compatibility matrices of a notional airship model.

```

% Prepare workspace
clc
close all
clear

addpath('..')

%% Morphological matrix
m = morphologicalMatrix('Airship');

r = rowConventional('Lifting medium');
r = r.addOption(option('Cold gas (Helium)', {}));
r = r.addOption(option('Hot air', {}));
m = m.addRow(r);

r = rowConventional('Empennage configuration');
r = r.addOption(option('Y', {}));
r = r.addOption(option('Inverted Y', {}));
r = r.addOption(option('X', {}));
r = r.addOption(option('Cross', {}));
m = m.addRow(r);

r = rowConventional('Ballonet-based pitch trim');
r = r.addOption(option('Yes', {}));
r = r.addOption(option('No', {}));
m = m.addRow(r);

r = rowConventional('Energy source');
r = r.addOption(option('Bio-chemical', {}));
r = r.addOption(option('Electric charge', {}));
r = r.addOption(option('Electrolyte', {}));
r = r.addOption(option('Hybrid', {}));
m = m.addRow(r);

r = rowConventional('Energy storage');
r = r.addOption(option('Battery (LiCoO2)', {}));
r = r.addOption(option('Battery (LiFePO4)', {}));
r = r.addOption(option('Battery (LiPo)', {}));
r = r.addOption(option('Battery (NiCad)', {}));
r = r.addOption(option('Battery (NiMH)', {}));
r = r.addOption(option('Fuel tank', {}));
r = r.addOption(option('External fuel tank', {}));
r = r.addOption(option('Electrolyte tank', {}));
r = r.addOption(option('Fuel tank + Battery (LiCoO2)', {}));
r = r.addOption(option('Fuel tank + Battery (LiFePO4)', {}));
r = r.addOption(option('Fuel tank + Battery (LiPo)', {}));
r = r.addOption(option('Fuel tank + Battery (NiCad)', {}));
r = r.addOption(option('Fuel tank + Battery (NiMH)', {}));

```

```

r = r.addOption(option('External fuel tank + Battery (LiCoO2)', {}));
r = r.addOption(option('External fuel tank + Battery (LiFePO4)', {}));
r = r.addOption(option('External fuel tank + Battery (LiPo)', {}));
r = r.addOption(option('External fuel tank + Battery (NiCad)', {}));
r = r.addOption(option('External fuel tank + Battery (NiMH)', {}));
m = m.addRow(r);

r = rowConventional('Converter to mechanical energy');
r = r.addOption(option('Piston', {}));
r = r.addOption(option('Turbine', {}));
r = r.addOption(option('Electric motor (Spinning cage, Brushed)', {}));
r = r.addOption(option('Electric motor (Spinning cage,
Brushless)', {}));
r = r.addOption(option('Electric motor (Spinning shaft, Brushed)', {}));
r = r.addOption(option('Electric motor (Spinning shaft,
Brushless)', {}));
r = r.addOption(option('Hybrid (Piston/Electric)', {}));
r = r.addOption(option('Hybrid (Turbine/Electric)', {}));
r = r.addOption(option('Fuel cell and electric motor', {}));
m = m.addRow(r);

r = rowConventional('Number of rotors');
r = r.addOption(option('1', {}));
r = r.addOption(option('2', {}));
r = r.addOption(option('3', {}));
r = r.addOption(option('4', {}));
r = r.addOption(option('6', {}));
r = r.addOption(option('8', {}));
m = m.addRow(r);

r = rowConventional('Steerable propulsion');
r = r.addOption(option('Yes', {}));
r = r.addOption(option('No', {}));
m = m.addRow(r);

r = rowConventional('Converter to lift/thrust');
r = r.addOption(option('Rotor', {}));
r = r.addOption(option('Fan', {}));
r = r.addOption(option('Propeller', {}));
m = m.addRow(r);

r = rowConventional('Number of blades');
r = r.addOption(option('2', {}));
r = r.addOption(option('3', {}));
r = r.addOption(option('4', {}));
r = r.addOption(option('6', {}));
r = r.addOption(option('8', {}));
r = r.addOption(option('10', {}));
m = m.addRow(r);

r = rowConventional('Blade type');
r = r.addOption(option('Fixed pitch', {}));
r = r.addOption(option('Variable pitch', {}));
m = m.addRow(r);

```

```

r = rowCombinatorial('Imaging');
r = r.addOption(option('Mono (Fixed-mount)', {}));
r = r.addOption(option('Mono (Gyro-stabilized)', {}));
r = r.addOption(option('RGB camera (Fixed-mount)', {}));
r = r.addOption(option('RGB camera (Gyro-stabilized)', {}));
r = r.addOption(option('Multispectral data (Fixed-mount)', {}));
r = r.addOption(option('Multispectral data (Gyro-stabilized)', {}));
r = r.addOption(option('Thermal camera (Fixed-mount)', {}));
r = r.addOption(option('Thermal camera (Gyro-stabilized)', {}));
m = m.addRow(r);

r = rowCombinatorial('Mapping');
r = r.addOption(option('2D LIDAR', {}));
r = r.addOption(option('3D LIDAR', {}));
r = r.addOption(option('Sonar', {}));
m = m.addRow(r);

r = rowCombinatorial('Attitude');
r = r.addOption(option('IMU+GPS', {}));
m = m.addRow(r);

r = rowCombinatorial('Altitude');
r = r.addOption(option('GPS', {}));
r = r.addOption(option('Barometer', {}));
r = r.addOption(option('Sonar', {}));
m = m.addRow(r);

r = rowConventional('Communications');
r = r.addOption(option('Radio', {}));
m = m.addRow(r);

r = rowConventional('Hull type');
r = r.addOption(option('Non-rigid', {}));
r = r.addOption(option('Semi-rigid', {}));
r = r.addOption(option('Rigid', {}));
m = m.addRow(r);

r = rowConventional('Battens');
r = r.addOption(option('Yes', {}));
r = r.addOption(option('No', {}));
m = m.addRow(r);

m = m.setUpCompatibilityIndices;
fprintf(m.toString)

%% Compatibility matrix
n = m.countOptions;

% Assume all compatible
c = m.initializeCompatibilityMatrix;
sum(sum(c == 0))

% Fill in incompatibilities
% Energy sources -----
---
```

```
c(9,13) = 0;  
c(9,14) = 0;  
c(9,15) = 0;  
c(9,16) = 0;  
c(9,17) = 0;  
c(9,20) = 0;  
c(9,21) = 0;  
c(9,22) = 0;  
c(9,23) = 0;  
c(9,24) = 0;  
c(9,25) = 0;  
c(9,26) = 0;  
c(9,27) = 0;  
c(9,28) = 0;  
c(9,29) = 0;  
c(9,30) = 0;  
  
c(9,33) = 0;  
c(9,34) = 0;  
c(9,35) = 0;  
c(9,36) = 0;  
c(9,37) = 0;  
c(9,38) = 0;  
c(9,39) = 0;  
  
c(10,18) = 0;  
c(10,19) = 0;  
c(10,20) = 0;  
c(10,21) = 0;  
c(10,22) = 0;  
c(10,23) = 0;  
c(10,24) = 0;  
c(10,25) = 0;  
c(10,26) = 0;  
c(10,27) = 0;  
c(10,28) = 0;  
c(10,29) = 0;  
c(10,30) = 0;  
  
c(10,31) = 0;  
c(10,32) = 0;  
c(10,37) = 0;  
c(10,38) = 0;  
c(10,39) = 0;  
  
c(11,13) = 0;  
c(11,14) = 0;  
c(11,15) = 0;  
c(11,16) = 0;  
c(11,17) = 0;  
c(11,18) = 0;  
c(11,19) = 0;  
c(11,21) = 0;  
c(11,22) = 0;  
c(11,23) = 0;  
c(11,24) = 0;
```

```

c(11,25) = 0;
c(11,26) = 0;
c(11,27) = 0;
c(11,28) = 0;
c(11,29) = 0;
c(11,30) = 0;

c(11,31) = 0;
c(11,32) = 0;
c(11,33) = 0;
c(11,34) = 0;
c(11,35) = 0;
c(11,36) = 0;
c(11,37) = 0;
c(11,38) = 0;

c(12,13) = 0;
c(12,14) = 0;
c(12,15) = 0;
c(12,16) = 0;
c(12,17) = 0;
c(12,18) = 0;
c(12,19) = 0;
c(12,20) = 0;

c(12,31) = 0;
c(12,32) = 0;
c(12,33) = 0;
c(12,34) = 0;
c(12,35) = 0;
c(12,36) = 0;
c(12,39) = 0;

% Energy storage -----
---
c(13,31) = 0; c(13,32) = 0; c(13,37) = 0; c(13,38) = 0; c(13,39) = 0;
c(14,31) = 0; c(14,32) = 0; c(14,37) = 0; c(14,38) = 0; c(14,39) = 0;
c(15,31) = 0; c(15,32) = 0; c(15,37) = 0; c(15,38) = 0; c(15,39) = 0;
c(16,31) = 0; c(16,32) = 0; c(16,37) = 0; c(16,38) = 0; c(16,39) = 0;
c(17,31) = 0; c(17,32) = 0; c(17,37) = 0; c(17,38) = 0; c(17,39) = 0;

c(18,33) = 0; c(18,34) = 0; c(18,35) = 0; c(18,36) = 0; c(18,37) = 0;
c(18,38) = 0; c(18,39) = 0;
c(19,33) = 0; c(19,34) = 0; c(19,35) = 0; c(19,36) = 0; c(19,37) = 0;
c(19,38) = 0; c(19,39) = 0;

c(20,31) = 0;
c(20,32) = 0;
c(20,33) = 0;
c(20,34) = 0;
c(20,35) = 0;
c(20,36) = 0;
c(20,37) = 0;
c(20,38) = 0;

```

```

for i = 21:30
    c(i,31) = 0;
    c(i,32) = 0;
    c(i,33) = 0;
    c(i,34) = 0;
    c(i,35) = 0;
    c(i,36) = 0;
    c(i,39) = 0;
end

% Hull type -----
---
c(75,78) = 0;

sum(sum(c == 0))

%% Symmetrize matrix
c = symmetrize(c);

% Total alternatives
m.computeAlternatives

figure
pcolor(c)
colormap(gray(2))
axis ij
axis square
shading flat
set(gca, 'FontName', 'Times New Roman', 'FontSize', 18)

% Compatible alternatives
tic
nn = m.computeCompatibleAlternatives(c, [], 0)
toc

```

Example output: airship.output.1805640.dedicated-sched.pace.gatech.edu

```

-----
Begin PBS Prologue Sun Nov 13 19:37:42 EST 2016
Job ID:      1805640.dedicated-sched.pace.gatech.edu
User ID:    jdurand7
Job name:   airship
Queue:     enterprise
End PBS Prologue Sun Nov 13 19:37:42 EST 2016
-----

          < M A T L A B (R) >
Copyright 1984-2016 The MathWorks, Inc.
R2016a (9.0.0.341360) 64-bit (glnxa64)
          February 11, 2016

```

To get started, type one of these: helpwin, helpdesk, or demo.
For product information, visit www.mathworks.com.

Academic License

Airship

Lifting medium

- (1) Cold gas (Helium) {}
- (2) Hot air {}

Empennage configuration

- (3) Y {}
- (4) Inverted Y {}
- (5) X {}
- (6) Cross {}

Ballonet-based pitch trim

- (7) Yes {}
- (8) No {}

Energy source

- (9) Bio-chemical {}
- (10) Electric charge {}
- (11) Electrolyte {}
- (12) Hybrid {}

Energy storage

- (13) Battery (LiCoO2) {}
- (14) Battery (LiFePO4) {}
- (15) Battery (LiPo) {}
- (16) Battery (NiCad) {}
- (17) Battery (NiMH) {}
- (18) Fuel tank {}
- (19) External fuel tank {}
- (20) Electrolyte tank {}
- (21) Fuel tank + Battery (LiCoO2) {}
- (22) Fuel tank + Battery (LiFePO4) {}
- (23) Fuel tank + Battery (LiPo) {}
- (24) Fuel tank + Battery (NiCad) {}
- (25) Fuel tank + Battery (NiMH) {}
- (26) External fuel tank + Battery (LiCoO2) {}
- (27) External fuel tank + Battery (LiFePO4) {}
- (28) External fuel tank + Battery (LiPo) {}
- (29) External fuel tank + Battery (NiCad) {}
- (30) External fuel tank + Battery (NiMH) {}

Converter to mechanical energy

- (31) Piston {}
- (32) Turbine {}
- (33) Electric motor (Spinning cage, Brushed) {}
- (34) Electric motor (Spinning cage, Brushless) {}
- (35) Electric motor (Spinning shaft, Brushed) {}
- (36) Electric motor (Spinning shaft, Brushless) {}
- (37) Hybrid (Piston/Electric) {}
- (38) Hybrid (Turbine/Electric) {}
- (39) Fuel cell and electric motor {}

Number of rotors

- (40) 1 {}
- (41) 2 {}
- (42) 3 {}

(43) 4 {}
(44) 6 {}
(45) 8 {}
Steerable propulsion
(46) Yes {}
(47) No {}
Converter to lift/thrust
(48) Rotor {}
(49) Fan {}
(50) Propeller {}
Number of blades
(51) 2 {}
(52) 3 {}
(53) 4 {}
(54) 6 {}
(55) 8 {}
(56) 10 {}
Blade type
(57) Fixed pitch {}
(58) Variable pitch {}
Imaging
(59) Mono (Fixed-mount) {}
(60) Mono (Gyro-stabilized) {}
(61) RGB camera (Fixed-mount) {}
(62) RGB camera (Gyro-stabilized) {}
(63) Multispectral data (Fixed-mount) {}
(64) Multispectral data (Gyro-stabilized) {}
(65) Thermal camera (Fixed-mount) {}
(66) Thermal camera (Gyro-stabilized) {}
Mapping
(67) 2D LIDAR {}
(68) 3D LIDAR {}
(69) Sonar {}
Attitude
(70) IMU+GPS {}
Altitude
(71) GPS {}
(72) Barometer {}
(73) Sonar {}
Communications
(74) Radio {}
Hull type
(75) Non-rigid {}
(76) Semi-rigid {}
(77) Rigid {}
Battens
(78) Yes {}
(79) No {}

ans =

242

ans =

```

441

ans =

3.3579e+11

nn =

111974400

Elapsed time is 8986.860981 seconds.
>> -----
Begin PBS Epilogue Sun Nov 13 22:07:40 EST 2016
Job ID:      1805640.dedicated-sched.pace.gatech.edu
User ID:    jdurand7
Job name:   airship
Resources:
neednodes=10:ppn=24,nodes=10:ppn=24,pmem=1024mb,walltime=05:00:00
Rsrc Used:
cput=02:29:51,energy_used=0,mem=508568kb,vmem=4772408kb,walltime=02:29:
58
Queue:      enterprise
Nodes:
rich133-q7-14-1.pace.gatech.edu rich133-q7-14-r.pace.gatech.edu
rich133-q7-15-1.pace.gatech.edu rich133-q7-15-r.pace.gatech.edu
rich133-q7-16-1.pace.gatech.edu rich133-q7-16-r.pace.gatech.edu
rich133-q7-17-1.pace.gatech.edu rich133-q7-17-r.pace.gatech.edu
rich133-q7-18-1.pace.gatech.edu rich133-q7-18-r.pace.gatech.edu
End PBS Epilogue Sun Nov 13 22:07:41 EST 2016
-----

```

B.3.6 Plots

A set of scripts used to generate the different figures and graphs seen in the design space exploration section (see page 325).

contourPlot.m

A contour plot showing the beneficial regions of morphological reduction.

```

% A Variable-Oriented Architecture Generation Methodology to Support
Efficient Multilevel Multidisciplinary Optimization
%
% Jean-Guillaume Durand (jdurand7@gatech.edu)

```

```

% Aerospace Systems Design Laboratory (ASDL)
% Georgia Institute of Technology, 2016

% Clean and prepare workspace
clc
close all
clear

% Problem parameters
nRows = 8;
nOptionsPerRow = 4;
nVars = 50;

kVec = 1:10:300; % Number of variables removed per option
nVec = 0:20; % Number of options removed

% Optimizer surrogate model
% Optimizer parameters
% Interpolate or use  $fc = @(n)-11.776*n.^2 + 5127.9*n + 13236;$ 
n = [1:5,10,50,100];
v = [14099.23077, 20625.53846, 25975.23077, 33926.46154, 38770.30769,
61857.38462, 178638.7692, 286491.3846];
fc = @(nq)interp1(n,v,nq,'linear','extrap');

%% Analysis
nk = length(kVec);
no = length(nVec);
nCallsOptimizer = zeros(nk, no);
for i = 1:nk
    for j = 1:no
        % Create morph matrix
        morph = nOptionsPerRow*ones(nRows,1);

        % Remove options
        counter = nVec(j);
        k = 1;
        while counter > 0
            % If there are options to be removed at this row
            if morph(k) > 1
                % Remove option
                morph(k) = morph(k) - 1;
                counter = counter - 1;
            else
                % Change row
                k = k + 1;
            end
        end

        % Compute total number of discrete alternatives
        nCallsOptimizer(i,j) = prod(morph);
    end
end

% Compute number of variables added to the optimizer
[X,Y] = meshgrid(nVec,kVec);

```

```

nVarsAdded = X.*Y;

% Compute number of function calls
nFunctionCalls = nCallsOptimizer .* fc(nVars + nVarsAdded);

Z = nFunctionCalls(1,1) - nFunctionCalls;

%% Plot
figure
surf(X,Y,-Z)

figure
contourf(X,Y,Z,100,'LineColor','none');
ax = gca;
ax.Position = [0.1300 0.1100 0.7 0.8150];
xlabel('Options removed','FontName','Times New Roman','FontSize',12)
ylabel('k','FontName','Times New Roman','FontSize',12)
set(gca,'FontName','Times New Roman','FontSize',12)

% Colormap
mymap = redgreencmap(64,'Interpolation','sigmoid');
mymap = mymap(end:-1:1,:); % Reverse order (from red to green)
colormap(mymap)

% Colobar
c = colorbar;
cmin = min(min(Z));
caxis([cmin -cmin]) % Scale
v = caxis;
c.Ticks = v(1):(v(2)-v(1))/6:v(2); % Divide in 6 segments
c.TickLabels = {'Most detrimental','Very
detrimental','Detrimental','Neutral','Beneficial','Very
beneficial','Most beneficial'};

```

influenceOfK.m

Influence of the k-factor on morphological reduction.

```

% A Variable-Oriented Architecture Generation Methodology to Support
Efficient Multilevel Multidisciplinary Optimization
%
% Jean-Guillaume Durand (jdurand7@gatech.edu)
% Aerospace Systems Design Laboratory (ASDL)
% Georgia Institute of Technology, 2016

% Clean and prepare workspace
clc
close all
clear

figure(1)

```

```

% b=axes('Position',[.1 .1 .8 1e-12]);
% set(b,'Units','normalized');
% set(b,'Color','none');
% set(b,'xlim',[nVarsAdded(1) nVarsAdded(end)]);

a=axes('Position',[.1 .2 .8 .7]);
set(a,'Units','normalized');
yyaxis(a,'left')

%% Initial problem size
% Morphological parameters
kVec = [5 25 100 300];
c = get(groot,'DefaultAxesColorOrder');
% c = hsv(length(kVec));

hPlain = zeros(1,length(kVec));
hDashed = zeros(1,length(kVec));
for m = 1:length(kVec)
nRows = 8;
nOptionsPerRow = 4;
k = kVec(m);

nVars = nRows*nOptionsPerRow*k;
nVars = 50;

% Optimizer parameters
% Interpolate or use fc = @(n)-11.776*n.^2 + 5127.9*n + 13236;
n = [1:5,10,50,100];
v = [14099.23077, 20625.53846, 25975.23077, 33926.46154, 38770.30769,
61857.38462, 178638.7692, 286491.3846];
fc = @(nq)interp1(n,v,nq,'linear','extrap');

fCalls = nOptionsPerRow^nRows * fc(nVars);

%% Analysis
nOptions = nOptionsPerRow*nRows;
nOptionsRemoved = 1:10;

% Effect on morphological decomposition
nCallsOptimizer = zeros(1, length(nOptionsRemoved));
for i = 1:length(nOptionsRemoved)
% Create morph matrix
morph = nOptionsPerRow*ones(nRows,1);

% Remove options
counter = nOptionsRemoved(i);
j = 1;
while counter > 0
% If there are options to be removed at this row
if morph(j) > 1
% Remove option
morph(j) = morph(j) - 1;
counter = counter - 1;
else
% Change row

```

```

        j = j + 1;
    end
end

% Compute total number of discrete alternatives
nCallsOptimizer(i) = prod(morph);
end

% Effect on optimizer
% Actual method
% Reduced number of options * more variables in optimizer
nVarsAdded = k*nOptionsRemoved;
fCallsAug = nCallsOptimizer .* fc(nVars + nVarsAdded);

% Plot
semilogy(nOptionsRemoved, fCallsAug, '-', 'color', c(m, :))
hold on
end

% Black baseline
semilogy(nOptionsRemoved, fCalls*ones(size(nOptionsRemoved,2),1), 'k-')

hold off

xlabel(a, 'Options removed', 'FontName', 'Times New Roman', 'FontSize', 12)
h = legend(...
    'k = 5', ...
    'k = 25', ...
    'k = 100', ...
    'k = 300', ...
    'Baseline');
set(h, 'FontName', 'Times New Roman', 'FontSize', 12)
set(a, 'FontName', 'Times New Roman', 'FontSize', 12)

ax = gca;
yyaxis left
ax.YColor = 'k';
aLim = ax.YLim;
aTick = ax.YTick;
ylabel(a, 'Function calls')
ax.FontName = 'Times New Roman';
ax.FontSize = 12;

yyaxis right
ax.YColor = 'k';
ax.YScale = 'log';
ax.YLim = aLim;
ax.YTick = [86400, 86400*7, 86400*30.5, 86400*365.25, 86400*365.25*10,
86400*365.25*100, 86400*365.25*1000];
ax.YTickLabel = {'1 day', '1 week', '1 month', '1 year', '10 years', '100
years', '1000 years'};

set(a, 'Position', [0.1 0.15 .75 .8])
a.XLim = [nOptionsRemoved(1), nOptionsRemoved(end)];

```

multilevel.m

Effects of morphological reduction on multiple levels of a design space.

```
% Design space computation
%
% Jean-Guillaume DURAND (jean-guillaume.durand@gatech.edu)
% Aerospace Systems Design Laboratory
% Georgia Institute of Technology
% 2016

%% Clean up
close all
clear variables
clc

% Model fit
% Interpolate or use  $fc = @(n)-11.776*n.^2 + 5127.9*n + 13236$ ;
n = [1:5,10,50,100];
v = [14099.23077, 20625.53846, 25975.23077, 33926.46154, 38770.30769,
61857.38462, 178638.7692, 286491.3846];
fc = @(nq)interp1(n,v,nq,'linear','extrap');
%fc = @(n)-11.776*n.^2 + 5127.9*n + 13236;
% fc = @(n)12614*n.^0.7484;

% Parameters
% Macroscopic level
n_rows_M = 5;
n_options_M = 3;

% Microscopic level
n_rows_m = 10;
n_options_m = 5;

% Initialization
n_options_total_m = n_options_m*n_rows_m;
n_options_total_M = n_options_M*n_rows_M;

%% Analysis
% Remove options at the microscopic level
n_alternatives_m = zeros(1,n_options_total_m+1);
for i = 0:n_options_total_m
    % Initialize morphological matrix
    morph = n_options_m*ones(n_rows_m,1);

    % Remove options
    j = 1;
    counter = i; % Number of options to be removed
    while counter > 0
        % If there are options to be removed at this row
        if morph(j) > 0
            % Remove option
            morph(j) = morph(j) - 1;
            counter = counter - 1;
        end
        j = j + 1;
    end
end
```

```

        else
            % Change row
            j = j + 1;
        end
    end
    disp(morph')
    % Remove rows with 0 options
    temp = morph(morph ~= 0);
    disp(temp')
    % Compute total number of discrete alternatives
    n_alternatives_m(i+1) = prod(temp);
    disp(prod(temp))
    fprintf('\n')
end

% Remove options at the macroscopic level
n_alternatives_M = zeros(1,n_options_total_M+1);
for i = 0:n_options_total_M
    % Initialize morphological matrix
    morph = n_options_M*ones(n_rows_M,1);

    % Remove options
    j = 1;
    counter = i; % Number of options to be removed
    while counter > 0
        % If there are options to be removed at this row
        if morph(j) > 0
            % Remove option
            morph(j) = morph(j) - 1;
            counter = counter - 1;
        else
            % Change row
            j = j + 1;
        end
    end
    disp(morph')
    % Remove rows with 0 options
    temp = morph(morph ~= 0);
    disp(temp')
    % Compute total number of discrete alternatives
    n_alternatives_M(i+1) = prod(temp);
    disp(prod(temp))
    fprintf('\n')
end

% Transfer corresponding variables to the optimizer
k = 3;
variables_initial_m = repmat(k*n_options_total_m, 1,
n_options_total_m+1);
variables_initial_M = repmat(k*n_options_total_M, 1,
n_options_total_M+1);

variables_added_m = k*(0:n_options_total_m);
variables_added_M = k*(0:n_options_total_M);

variables_total_m = variables_initial_m + variables_added_m;

```



```

variables_total_M = variables_initial_M + variables_added_M;

fcalls_m = n_alternatives_m .* fc(variables_total_m);
fcalls_M = n_alternatives_M .* fc(variables_total_M);

%% Plot 1
figure
subplot(1,2,1)
semilogy(0:n_options_total_m, fcalls_m)
xlabel('Options removed','FontName','Times New Roman','FontSize',12)
ylabel('Function calls','FontName','Times New Roman','FontSize',12)
set(gca,'FontName','Times New Roman','FontSize',12)
title('Microscopic')
xlim([0,n_options_total_m])

%% Plot 2
subplot(1,2,2)
semilogy(0:n_options_total_M, fcalls_M)
xlabel('Options removed','FontName','Times New Roman','FontSize',12)
ylabel('Function calls','FontName','Times New Roman','FontSize',12)
set(gca,'FontName','Times New Roman','FontSize',12)
title('Macroscopic')

%% Plot 3
[X,Y] = meshgrid(0:n_options_total_M,0:n_options_total_m);
[Xvar,Yvar] = meshgrid(variables_total_M,variables_total_m);
Nvar = Xvar + Yvar;
% Z = fcalls_m'*fcalls_M;
Z = n_alternatives_m'*n_alternatives_M.*fc(Nvar);
figure
surf(X,Y,log10(Z))
caxis(log10([min(min(Z)), max(max(Z))]))
xlabel({'Macroscopic', 'options removed'},'FontName','Times New Roman','FontSize',12)
ylabel({'Microscopic', 'options removed'},'FontName','Times New Roman','FontSize',12)
zlabel('Function calls','FontName','Times New Roman','FontSize',12)
set(gca,'FontName','Times New Roman','FontSize',12)

% Create logarithmic axis
ZTick = 0:5:50;
ZTickLabels = cellstr(num2str(round(ZTick(:)), '10^{%d}'));
set(gca,'ztick',ZTick)
set(gca,'zticklabel',ZTickLabels)

% Create logarithmic color bar
c = colorbar;
cLimits = c.Limits;
cTicks = round(cLimits(1)):1:round(cLimits(2));
c.Ticks = cTicks;
c.TickLabels = cellstr(num2str(cTicks(:), '10^{%d}'));
c.FontSize = 12;

%% Plot 4
figure

```

```

subplot(1,2,2)
plot([0 1 2], [Z(2,1) Z(2,2) Z(2,3)], 'LineWidth',2)
YLim = get(gca, 'YLim');
hold on
plot([1 1], YLim, 'r--')
plot([0 2], [Z(2,2) Z(2,2)], 'r--')
plot(1, Z(2,2), 'r*')
hold off
xlabel('Options removed', 'FontName', 'Times New Roman', 'FontSize', 12)
ylabel('Function calls', 'FontName', 'Times New Roman', 'FontSize', 12)
title('Macroscopic')
set(gca, 'XMinorTick', 'on', 'YMinorTick', 'on')
set(gca, 'FontName', 'Times New Roman', 'FontSize', 12)

subplot(1,2,1)
plot([0 1 2], [Z(1,2) Z(2,2) Z(3,2)], 'LineWidth',2)
hold on
plot([1 1], YLim, 'r--')
plot([0 2], [Z(2,2) Z(2,2)], 'r--')
plot(1, Z(2,2), 'r*')
hold off
xlabel('Options removed', 'FontName', 'Times New Roman', 'FontSize', 12)
ylabel('Function calls', 'FontName', 'Times New Roman', 'FontSize', 12)
title('Microscopic')
set(gca, 'FontName', 'Times New Roman', 'FontSize', 12)
set(gca, 'XMinorTick', 'on', 'YMinorTick', 'on')
set(gca, 'YLim', YLim)

```

strategyDifference.m

A plot showing how the choice between “rows first” and “columns first” strategies affects the behavior of morphological reduction.

```

% A Variable-Oriented Architecture Generation Methodology to Support
Efficient Multilevel Multidisciplinary Optimization
%
% Jean-Guillaume Durand (jdurand7@gatech.edu)
% Aerospace Systems Design Laboratory (ASDL)
% Georgia Institute of Technology, 2016

% Clean and prepare workspace
clc
close all
clear

%% Initialize plot
figure(1)

a=axes('Position',[.1 .2 .8 .7]);
set(a, 'Units', 'normalized');

```

```

yyaxis(a, 'left')

%% Plot 1
% Morphological parameters
nRows = 8;
nOptionsPerRow = 4;
k = 3;

nVars = nRows*nOptionsPerRow*k;

% Optimizer parameters
% Interpolate or use fc = @(n)-11.776*n.^2 + 5127.9*n + 13236;
n = [1:5,10,50,100];
v = [14099.23077, 20625.53846, 25975.23077, 33926.46154, 38770.30769,
61857.38462, 178638.7692, 286491.3846];
fc = @(nq)interp1(n,v,nq, 'linear', 'extrap');

fCalls = nOptionsPerRow^nRows * fc(nVars);

% Analysis
nOptionsRemoved = 1:21;

% Effect on morphological decomposition
nCallsOptimizer = zeros(1, length(nOptionsRemoved));
for i = 1:length(nOptionsRemoved)
    % Create morph matrix
    morph = nOptionsPerRow*ones(nRows,1);

    % Remove options
    counter = nOptionsRemoved(i);
    j = 1;
    while counter > 0
        % If there are options to be removed at this row
        if morph(j) > 0
            % Remove option
            morph(j) = morph(j) - 1;
            counter = counter - 1;
        else
            % Change row
            j = j + 1;
        end
    end
    disp(morph')
    % Remove rows with 0 options
    temp = morph(morph ~= 0);
    disp(temp')
    % Compute total number of discrete alternatives
    nCallsOptimizer(i) = prod(temp);
    disp(prod(temp))
    fprintf('\n')
end
morph'

% Effect on optimizer
% Actual method

```

```

% Reduced number of options * more variables in optimizer
nVarsAdded = k*nOptionsRemoved;
fCallsAug = nCallsOptimizer .* fc(nVars + nVarsAdded);

% Plot
semilogy(nOptionsRemoved, fCalls*ones(size(nOptionsRemoved,2),1), 'k-')
hold on
semilogy(nOptionsRemoved, fCallsAug, 'b-')

%% Plot 2
% Morphological parameters
nRows = 8;
nOptionsPerRow = 4;
k = 3;

nVars = nRows*nOptionsPerRow*k;

% Optimizer parameters
% Interpolate or use fc = @(n)-11.776*n.^2 + 5127.9*n + 13236;
n = [1:5,10,50,100];
v = [14099.23077, 20625.53846, 25975.23077, 33926.46154, 38770.30769,
61857.38462, 178638.7692, 286491.3846];
fc = @(nq)interp1(n,v,nq, 'linear', 'extrap');

fCalls = nOptionsPerRow^nRows * fc(nVars);

% Analysis
nOptions = nOptionsPerRow*nRows;
nOptionsRemoved = 1:21;

% Effect on morphological decomposition
nCallsOptimizer = zeros(1, length(nOptionsRemoved));
for i = 1:length(nOptionsRemoved)
    % Create morph matrix
    morph = nOptionsPerRow*ones(nRows,1);

    % Remove options
    counter = nOptionsRemoved(i);
    j = 1;
    while counter > 0
        % If there are options to be removed at this row
        if morph(j) > 0
            % Remove option
            morph(j) = morph(j) - 1;
            counter = counter - 1;
        %
        %
        else
            j = j+1;
        end

        % Change row
        j = mod(j + 1, length(morph));
        if j == 0
            j = length(morph);
        end
    end
end
end

```

```

% Remove rows with 0 options
temp = morph(morph ~= 0);

% Compute total number of discrete alternatives
nCallsOptimizer(i) = prod(temp);
end
morph'

% Effect on optimizer
% Actual method
% Reduced number of options * more variables in optimizer
nVarsAdded = k*nOptionsRemoved;
fCallsAug = nCallsOptimizer .* fc(nVars + nVarsAdded);

% Plot
semilogy(nOptionsRemoved, fCallsAug, 'r-')

hold off

%% Finish plot formatting
xlabel(a, 'Options removed', 'FontName', 'Times New Roman', 'FontSize', 12)
h = legend(...
    'Baseline', ...
    'Rows first', ...
    'Columns first');
set(h, 'FontName', 'Times New Roman', 'FontSize', 12)
set(a, 'FontName', 'Times New Roman', 'FontSize', 12)

ax = gca;
yyaxis left
ax.YColor = 'k';
aLim = ax.YLim;
aTick = ax.YTick;
ylabel(a, 'Function calls')
ax.FontName = 'Times New Roman';
ax.FontSize = 12;

yyaxis right
ax.YColor = 'k';
ax.YScale = 'log';
ax.YLim = aLim;
ax.YTick = [86400, 86400*7, 86400*30.5, 86400*365.25, 86400*365.25*10,
86400*365.25*100, 86400*365.25*1000];
ax.YTickLabel = {'1 day', '1 week', '1 month', '1 year', '10 years', '100
years', '1000 years'};

set(a, 'Position', [0.1 0.15 .75 .8])
a.XLim = [nOptionsRemoved(1), nOptionsRemoved(end)];

```

tradeoff.m

A plot displaying the tradeoff between the classical approach and morphological reduction.

```
% A Variable-Oriented Architecture Generation Methodology to Support
Efficient Multilevel Multidisciplinary Optimization
%
% Jean-Guillaume Durand (jdurand7@gatech.edu)
% Aerospace Systems Design Laboratory (ASDL)
% Georgia Institute of Technology, 2016

% Clean and prepare workspace
clc
close all
clear

%% Initial problem size
% Morphological parameters
nRows = 8;
nOptionsPerRow = 4;
k = 5; % 100
% TODO - need to consider a different number of variables initially

nVars = nRows*nOptionsPerRow*k;
nVars = 50;

% Optimizer parameters
% Interpolate or use  $fc = @(n)-11.776*n.^2 + 5127.9*n + 13236;$ 
n = [1:5,10,50,100];
v = [14099.23077, 20625.53846, 25975.23077, 33926.46154, 38770.30769,
61857.38462, 178638.7692, 286491.3846];
fc = @(nq)interp1(n,v,nq,'linear','extrap');

fCalls = nOptionsPerRow^nRows * fc(nVars);

%% Analysis
nOptionsRemoved = 1:10;

% Effect on morphological decomposition
nCallsOptimizer = zeros(1, length(nOptionsRemoved));
for i = 1:length(nOptionsRemoved)
    % Create morph matrix
    morph = nOptionsPerRow*ones(nRows,1);

    % Remove options
    counter = nOptionsRemoved(i);
    j = 1;
    while counter > 0
        % If there are options to be removed at this row
        if morph(j) > 1
            % Remove option
            morph(j) = morph(j) - 1;
            counter = counter - 1;
        else
            % Change row
            j = j + 1;
        end
    end
end
```

```

        end
    end

    % Compute total number of discrete alternatives
    nCallsOptimizer(i) = prod(morph);
end

% Effect on optimizer
% Actual method
% Reduced number of options * more variables in optimizer
nVarsAdded = k*nOptionsRemoved;
fCallsAug = nCallsOptimizer .* fc(nVars + nVarsAdded);

% What if optimizer took the same time?
fCallsAug1 = nCallsOptimizer .* fc(nVars);

% What if constant number of discrete calls?
fCallsAug2 = nOptionsPerRow^nRows .* fc(nVars + nVarsAdded);

%% Plot
figure(1)

b=axes('Position',[.1 .1 .8 1e-12]);
set(b,'Units','normalized');
set(b,'Color','none');
set(b,'xlim',[nVarsAdded(1) nVarsAdded(end)]);

a=axes('Position',[.1 .2 .8 .7]);
set(a,'Units','normalized');
yyaxis(a,'left')

semilogy(nOptionsRemoved, fCalls*ones(size(nOptionsRemoved,2),1), 'b')

hold on
semilogy(nOptionsRemoved, fCallsAug, 'r')
semilogy(nOptionsRemoved, fCallsAug1, 'r--')
semilogy(nOptionsRemoved, fCallsAug2, 'b--')
% hold off

xlabel(a, 'Options removed', 'FontName', 'Times New Roman', 'FontSize', 12)
xlabel(b, 'Variables added', 'FontName', 'Times New Roman', 'FontSize', 12)
h = legend(...
    'Baseline',...
    'Morphological reduction',...
    'Constant number of variables',...
    'Constant number of options');
set(h, 'FontName', 'Times New Roman', 'FontSize', 12)
set(a, 'FontName', 'Times New Roman', 'FontSize', 12)
set(b, 'FontName', 'Times New Roman', 'FontSize', 12)

ax = gca;
yyaxis left
ax.YColor = 'k';
aLim = ax.YLim;
aTick = ax.YTick;

```

```

ylabel(a, 'Function calls')
ax.FontName = 'Times New Roman';
ax.FontSize = 12;

yyaxis right
ax.YColor = 'k';
ax.YScale = 'log';
ax.YLim = aLim;
ax.YTick = [86400*365.25, 86400*365.25*10, 86400*365.25*100,
86400*365.25*1000];
ax.YTickLabel = {'1 Year', '10 years', '100 years', '1000 years'};

% Adjust scale so that we can see the axes labels properly
set(a, 'Position', [0.1 0.25 .75 .65])
set(b, 'Position', [0.1 0.12 .75 1e-12])
a.XLim = [nOptionsRemoved(1), nOptionsRemoved(end)];

% Durations in seconds
durations = [1, 60, 3600, 86400, 86400*7, 86400*30.5, 86400*365.25,
86400*365.25*[10 100 1000]];

```

tradeoff2parameters.m

Similar tradeoff study including different options removal strategies.

```

% A Variable-Oriented Architecture Generation Methodology to Support
Efficient Multilevel Multidisciplinary Optimization
%
% Jean-Guillaume Durand (jdurand7@gatech.edu)
% Aerospace Systems Design Laboratory (ASDL)
% Georgia Institute of Technology, 2016

% Clean and prepare workspace
clc
close all
clear

%% Initialize plot
figure(1)

% b=axes('Position',[.1 .1 .8 1e-12]);
% set(b,'Units','normalized');
% set(b,'Color','none');
% set(b,'xlim',[nVarsAdded(1) nVarsAdded(end)]);

a=axes('Position',[.1 .2 .8 .7]);
set(a,'Units','normalized');
yyaxis(a,'left')

%% Plot 1
% Morphological parameters
nRows = 8;

```



```

nOptionsPerRow = 3;
k = 4;

nVars = nRows*nOptionsPerRow*k;

% Optimizer parameters
% Interpolate or use fc = @(n)-11.776*n.^2 + 5127.9*n + 13236;
n = [1:5,10,50,100];
v = [14099.23077, 20625.53846, 25975.23077, 33926.46154, 38770.30769,
61857.38462, 178638.7692, 286491.3846];
fc = @(nq)interp1(n,v,nq,'linear','extrap');

fCalls = nOptionsPerRow^nRows * fc(nVars);

% Analysis
nOptionsRemoved = 1:15;

% Effect on morphological decomposition
nCallsOptimizer = zeros(1, length(nOptionsRemoved));
for i = 1:length(nOptionsRemoved)
    % Create morph matrix
    morph = nOptionsPerRow*ones(nRows,1);

    % Remove options
    counter = nOptionsRemoved(i);
    j = 1;
    while counter > 0
        % If there are options to be removed at this row
        if morph(j) > 1
            % Remove option
            morph(j) = morph(j) - 1;
            counter = counter - 1;
        else
            % Change row
            j = j + 1;
        end
    end

    % Compute total number of discrete alternatives
    nCallsOptimizer(i) = prod(morph);
end

% Effect on optimizer
% Actual method
% Reduced number of options * more variables in optimizer
nVarsAdded = k*nOptionsRemoved;
fCallsAug = nCallsOptimizer .* fc(nVars + nVarsAdded);

% Plot
h0 = semilogy(nOptionsRemoved,
fCalls*ones(size(nOptionsRemoved,2),1),'w');
hold on
hBlack(1) = semilogy(nOptionsRemoved,
fCalls*ones(size(nOptionsRemoved,2),1),'k-');
hBlack(2) = semilogy(nOptionsRemoved, fCallsAug,'k--');

```

```

hPlot(1) = semilogy(nOptionsRemoved,
fCalls*ones(size(nOptionsRemoved,2),1), 'b-');
semilogy(nOptionsRemoved, fCallsAug, 'b--')

%% Plot 2
% Morphological parameters
nRows = 8;
nOptionsPerRow = 3;
k = 4;

nVars = nRows*nOptionsPerRow*k;

% Optimizer parameters
% Interpolate or use fc = @(n)-11.776*n.^2 + 5127.9*n + 13236;
n = [1:5,10,50,100];
v = [14099.23077, 20625.53846, 25975.23077, 33926.46154, 38770.30769,
61857.38462, 178638.7692, 286491.3846];
fc = @(nq)interp1(n,v,nq, 'linear', 'extrap');

fCalls = nOptionsPerRow^nRows * fc(nVars);

% Analysis
nOptions = nOptionsPerRow*nRows;
nOptionsRemoved = 1:15;

% Effect on morphological decomposition
nCallsOptimizer = zeros(1, length(nOptionsRemoved));
for i = 1:length(nOptionsRemoved)
    % Create morph matrix
    morph = nOptionsPerRow*ones(nRows,1);

    % Remove options
    counter = nOptionsRemoved(i);
    j = 1;
    while counter > 0
        % If there are options to be removed at this row
        if morph(j) > 1
            % Remove option
            morph(j) = morph(j) - 1;
            counter = counter - 1;
        %
        %
        else
            j = j+1;
        end

        % Change row
        j = mod(j + 1, length(morph));
        if j == 0
            j = length(morph);
        end
    end

    % Compute total number of discrete alternatives
    nCallsOptimizer(i) = prod(morph);
end

```

```

% Effect on optimizer
% Actual method
% Reduced number of options * more variables in optimizer
nVarsAdded = k*nOptionsRemoved;
fCallsAug = nCallsOptimizer .* fc(nVars + nVarsAdded);

% Plot
hPlot(2) = semilogy(nOptionsRemoved,
fCalls*ones(size(nOptionsRemoved,2),1), 'r-');
semilogy(nOptionsRemoved, fCallsAug, 'r--')

% Black line
semilogy(nOptionsRemoved, fCalls*ones(size(nOptionsRemoved,2),1), 'k-')

hold off

%% Finish plot formatting
xlabel(a, 'Options removed', 'FontName', 'Times New Roman', 'FontSize', 12)
h = legend([hPlot, h0, hBlack(1), hBlack(2)], ...
    'Rows first', ...
    'Columns first', ...
    '', ...
    'Baseline', ...
    'Morphological reduction');
set(h, 'FontName', 'Times New Roman', 'FontSize', 12)
set(a, 'FontName', 'Times New Roman', 'FontSize', 12)

ax = gca;
yyaxis left
ax.YColor = 'k';
aLim = ax.YLim;
aTick = ax.YTick;
ylabel(a, 'Function calls')
ax.FontName = 'Times New Roman';
ax.FontSize = 12;

yyaxis right
ax.YColor = 'k';
ax.YScale = 'log';
ax.YLim = aLim;
ax.YTick = [86400, 86400*7, 86400*30.5, 86400*365.25, 86400*365.25*10,
86400*365.25*100, 86400*365.25*1000];
ax.YTickLabel = {'1 day', '1 week', '1 month', '1 year', '10 years', '100
years', '1000 years'};

set(a, 'Position', [0.1 0.15 .75 .8])
a.XLim = [nOptionsRemoved(1), nOptionsRemoved(end)];

```

APPENDIX C

ROS/GAZEBO FILES

The ROS/Gazebo files regroup the set of files necessary to setup the simulation using the microscopic modeling approach (see section 4.2.3 page 266). They are organized into three ROS packages, each with a specific purpose:

- **gritbot_description:** pure ROS description of the Gritbot robot.
- **gritbot_gazebo:** wrappers for the Gazebo simulator to ensure the link with ROS.
- **gritbot_navigation:** intelligence of the robots.

It is highly recommended for the reader to consult the ROS tutorials in order to understand where to include each file in the set of created packages. An easy way to re-use the same files is to create three packages and follow the tutorials to create a robot, and then replace the proper files with the ones included here below. A few additional libraries are required to be linked to these packages and are described in the microscopic modeling section.

Note that to properly setup the simulator, the Gritbots design files should be downloaded and converted to proper meshes compatible with Gazebo. Blender and its addons is a potential tool for such a task. In particular, if the computer vision trackers are used, each robot must have a COLLADA (file extension *.dae*) description file which refers to a unique texture encoding the ID tag.

C.1 Gritbot URDF description file

In total, 20 Gritbot description files were created, each having a different Aruco tag, this Aruco tag is the only change in each of the files. A typical example of file is given here below.

gritbot_1.urdf

```
<robot name="gritbot">
  <!-- the model -->
  <link name="base_link">
    <inertial>
      <mass value="0.05" />
      <origin xyz="0 0 0" />
      <inertia ixx="0.000075" ixy="0.0" ixz="0.0"
        iyy="0.000075" iyz="0.0"
        izz="0.000075" />
    </inertial>
    <visual>
      <geometry>
        <mesh
filename="package://gritbot_description/meshes/gritbot_without_shell.dae" />
          <!-- <box size=".03 .03 .03" /> -->
        </geometry>
      </visual>
      <collision>
        <origin xyz="0 0 0.0036" />
        <geometry>
          <box size="0.031 0.045 0.033" />
        </geometry>
      </collision>
    </link>

    <link name="apriltag_link">
      <visual>
        <geometry>
          <mesh
filename="package://gritbot_description/meshes/aruco_tag_1.dae" />
            <box size=".03 .03 .01" />
          </geometry>
        </visual>
        <collision>
          <origin xyz="0 0 0" />
          <geometry>
            <box size=".03 .03 .01" />
          </geometry>
        </collision>
      </link>

    <joint name="base_to_apriltag" type="fixed">
      <parent link="base_link"/>
```

```

    <child link="apriltag_link"/>
    <origin xyz="0 0 .0216"/>
</joint>

<!-- root link, on the ground just below the model origin -->
<link name="base_footprint">
  <visual>
    <origin xyz="0 0 0" rpy="0 0 0" />
    <geometry>
      <box size="0.001 0.001 0.001" />
    </geometry>
  </visual>
</link>

<joint name="base_link_joint" type="fixed">
  <origin xyz="0 0 0" rpy="0 0 0" />
  <parent link="base_footprint"/>
  <child link="base_link" />
</joint>

<gazebo>
  <plugin name="object_controller"
filename="libgazebo_ros_planar_move.so">
    <commandTopic>cmd_vel</commandTopic>
    <odometryTopic>odom</odometryTopic>
    <odometryFrame>odom</odometryFrame>
    <odometryRate>20.0</odometryRate>
    <robotBaseFrame>base_footprint</robotBaseFrame>
  </plugin>
</gazebo>

  <!-- TODO, add 6 IR sensors, accelerometer, gyroscope, battery
voltage sensor -->
  <!-- Set frequencies and queue lengths correctly in ROSnodes -->
</robot>

```

C.2 *Robotarium world file*

This file is used by Gazebo to generate a world where the robots and the mission will be simulated. This world consists for the most part of the Robotarium arena. The robots and the camera tracker are spawned separately as world add-ons thanks to the launch file.

robotarium.world

```

<?xml version="1.0" ?>
<sdf version="1.4">
  <world name="default">
    <include>
      <uri>model://ground_plane</uri>
    </include>

```

```

<include>
  <uri>model://sun</uri>
</include>
<gravity>0 0 -9.81</gravity>
<model name="left">
  <pose>-.6 0 0 0 0 0</pose>
  <link name="link">
    <collision name="collision">
      <geometry>
        <box>
          <size>.01 .7 .05</size>
        </box>
      </geometry>
    </collision>
    <visual name="visual">
      <geometry>
        <box>
          <size>.01 .7 .05</size>
        </box>
      </geometry>
      <material>
        <script>Gazebo/Blue</script>
      </material>
    </visual>
  </link>
</model>
<model name="right">
  <pose>.6 0 0 0 0 0</pose>
  <link name="link">
    <collision name="collision">
      <geometry>
        <box>
          <size>.01 .7 .05</size>
        </box>
      </geometry>
    </collision>
    <visual name="visual">
      <geometry>
        <box>
          <size>.01 .7 .05</size>
        </box>
      </geometry>
      <material>
        <script>Gazebo/Blue</script>
      </material>
    </visual>
  </link>
</model>
<model name="bottom">
  <pose>0 -.35 0 0 0 0</pose>
  <link name="link">
    <collision name="collision">
      <geometry>
        <box>
          <size>1.2 .01 .05</size>
        </box>
      </geometry>

```

```

        </collision>
        <visual name="visual">
            <geometry>
                <box>
                    <size>1.2 .01 .05</size>
                </box>
            </geometry>
            <material>
                <script>Gazebo/Blue</script>
            </material>
        </visual>
    </link>
</model>
<model name="top">
    <pose>0 .35 0 0 0 0</pose>
    <link name="link">
        <collision name="collision">
            <geometry>
                <box>
                    <size>1.2 .01 .05</size>
                </box>
            </geometry>
        </collision>
        <visual name="visual">
            <geometry>
                <box>
                    <size>1.2 .01 .05</size>
                </box>
            </geometry>
            <material>
                <script>Gazebo/Blue</script>
            </material>
        </visual>
    </link>
</model>
</world>
</sdf>

```

C.3 Consensus mission launch file

The launch file is the starting point for the microscopic simulation. Once the *roscore* is running in a terminal, one can simply open a new terminal and *roslaunch* the file *robotarium.launch* to start the simulation of the rendezvous mission. Note that the launch file is modified based on the number of robots to be included in the mission as well as their initial positions (see the Matlab generation files in APPENDIX B). Finally, the

commented part of the code refers to another possible computer vision package used to track the ID tags of the robots.

robotarium.launch

```
<?xml version="1.0"?>
<launch>
  <!-- Environment
#####
#####
#####-->
  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="world_name" value="$(find
gritbot_gazebo)/worlds/robotarium.world"/>
    <arg name="paused" value="false"/>
    <arg name="use_sim_time" value="true"/>
    <arg name="gui" value="false"/>
    <arg name="headless" value="true"/>
    <arg name="debug" value="false"/>
  </include>

  <!-- Robots
#####
#####
#####-->
  <node name="gritbot1" ns="gritbot1" pkg="gazebo_ros"
type="spawn_model" args="-file $(find
gritbot_description)/urdf/gritbot_1.urdf -urdf -x -.25 -y -.25 -z 0 -
model gritbot1" />
  <node name="gritbot2" ns="gritbot2" pkg="gazebo_ros"
type="spawn_model" args="-file $(find
gritbot_description)/urdf/gritbot_2.urdf -urdf -x .25 -y -.25 -z 0 -
model gritbot2" />
  <node name="gritbot3" ns="gritbot3" pkg="gazebo_ros"
type="spawn_model" args="-file $(find
gritbot_description)/urdf/gritbot_3.urdf -urdf -x .25 -y .25 -z 0 -
model gritbot3" />
  <node name="gritbot4" ns="gritbot4" pkg="gazebo_ros"
type="spawn_model" args="-file $(find
gritbot_description)/urdf/gritbot_4.urdf -urdf -x -.25 -y .25 -z 0 -
model gritbot4" />

  <!-- Robotarium tracker
#####
#####
##### -->
  <!-- Camera -->
  <node name="tracker" ns="robotarium" pkg="gritbot_navigation"
type="tracker" />
  <node name="camera" ns="robotarium" pkg="gazebo_ros"
type="spawn_model" args="-file $(find
gritbot_description)/urdf/camera.urdf -urdf -model camera" />
  <node name="camera_view" pkg="image_view" type="image_view">
    <remap from="image" to="/robotarium/camera/image_raw" />
  </node>
</launch>
```

```

        <param name="approximate_sync" value="true" />
    </node>

    <!-- Undistort images -->
    <node name="image_proc" ns="/robotarium/camera" pkg="image_proc"
type="image_proc" />
    <node name="corrected_view" pkg="image_view" type="image_view">
        <remap from="image"
to="/robotarium/camera/image_rect_color" />
        <param name="approximate_sync" value="true" />
    </node>

    <!-- Track Aruco Tags -->
    <node pkg="aruco_ros" type="single" name="aruco_single">
        <remap from="/camera_info" to="/robotarium/camera/camera_info"
/>
        <remap from="/image" to="/robotarium/camera/image_rect_color"
/>

        <param name="image_is_rectified" value="True"/>
        <param name="marker_size" value="0.032"/>
        <param name="marker_id" value="1"/>
        <param name="reference_frame" value=""/> <!-- frame in
which the marker pose will be refered -->
        <param name="camera_frame" value="base_footprint"/>
        <param name="marker_frame" value="base_footprint" />
        <param name="corner_refinement" value="LINES" />
    </node>
    <node name="aruco_view" pkg="image_view" type="image_view">
        <remap from="image" to="/aruco_single/result" />
        <param name="approximate_sync" value="true" />
    </node>

    <!--<arg name="marker_size" default="4.4" />
    <arg name="max_new_marker_error" default="0.08" />
    <arg name="max_track_error" default="0.2" />
    <arg name="cam_image_topic" default="/robotarium/camera/image_raw"
/>
    <arg name="cam_info_topic" default="/robotarium/camera/camera_info"
/>
    <arg name="output_frame" default="/camera/base_link" />

    <node name="ar_track_alvar" pkg="ar_track_alvar"
type="individualMarkersNoKinect" respawn="false" output="screen"
args="$(arg marker_size) $(arg max_new_marker_error) $(arg
max_track_error) $(arg cam_image_topic) $(arg cam_info_topic) $(arg
output_frame)" />
-->
    <!-- Intelligence
#####
#####
##### -->
    <!-- Data logger -->
    <node name="logger" ns="robotarium" pkg="gritbot_navigation"
type="logger" args="/home/jdurand7/logs/log_4_0.1.csv" />

    <!-- Static consensus algorithm -->

```

```
<!-- NOTE: this node is required, if it stops, the whole simulation
stops -->
<!--<node name="consensus" ns="robotarium" pkg="gritbot_navigation"
type="consensus" args="0.1" output="screen" required="true"/>-->
</launch>
```

C.4 Navigation package

As opposed to the previous packages *gritbot_description* and *gritbot_gazebo* which encompass the physical description and simulation of the robots, the *navigation* package represents the intelligence of the robot and is the one to read the sensors inputs, run the consensus mission, and output the results. More details about the implementation can be found in section 4.2.3 page 266.

C.4.1 Consensus

This class is designed to be instantiated with a given number of robots and their individual velocity, and perform a static consensus for these robots. In particular, it is in constant communication with the Gazebo simulator to read the location of the robots derived from the tracking system, compute appropriate velocities for the robots to meet at consensus without any collisions, and return these velocities as orders to the controllers for the robots in the simulator.

consensus.cpp

```
#include <gazebo_msgs/ModelState.h>
#include <ros/ros.h>
#include <tf/transform_datatypes.h>
#include <geometry_msgs/Twist.h>
#include <vector>

#include <CGAL/QP_functions.h>
#include <CGAL/MP_Float.h>

typedef CGAL::MP_Float ET;
typedef CGAL::Quadratic_program<ET> Program;
typedef CGAL::Quadratic_program_solution<ET> Solution;

class RobotariumConsensus
```

```

{
public:
    // Constructor
    RobotariumConsensus();
    void setMaxLinearVelocity(double maxLinearVelocity);

private:
    // Attributes
    int count;
    bool init;
    double v;
    ros::NodeHandle nh_;
    ros::Subscriber poses_sub_;
    ros::Publisher poses_pub_;
    std::vector<ros::Publisher> myvector;
    std::vector<double> previous_x;
    std::vector<double> previous_y;

    // Methods
    int nChooseK(int iN, int iR);
    std::vector<int> topologicalNeighbors(int i, int N);
    std::vector<geometry_msgs::Twist>
    saturateVelocity(std::vector<geometry_msgs::Twist> dxi, double
    maxLinear, double maxAngular);
    std::vector<geometry_msgs::Twist>
    barrierCertificate(std::vector<geometry_msgs::Twist> dxi, const
    gazebo_msgs::ModelState::ConstPtr& msg, double gamma, double
    safetyRadius);
    gazebo_msgs::ModelState::ConstPtr uniToSiStates(const
    gazebo_msgs::ModelState::ConstPtr& msg, double projectionDistance);
    std::vector<geometry_msgs::Twist>
    int2uni3(std::vector<geometry_msgs::Twist> dxi, const
    gazebo_msgs::ModelState::ConstPtr& msg, double lambda);
    double computeAverageDistanceChange(const
    gazebo_msgs::ModelState::ConstPtr& msg);
    geometry_msgs::Twist computeRobotVelocity(int robotID, const
    geometry_msgs::Pose pose, const gazebo_msgs::ModelState::ConstPtr&
    msg, double projectionDistance);
    void poseCallback(const gazebo_msgs::ModelState::ConstPtr& msg);
};

RobotariumConsensus::RobotariumConsensus():
    init(true),
    count(0),
    v(.1) // m/s
{
    // Declare topics
    poses_sub_ = nh_.subscribe("/robotarium/poses", 1,
    &RobotariumConsensus::poseCallback, this);
    poses_pub_ =
    nh_.advertise<gazebo_msgs::ModelState>("/robotarium/controller/poses",
    1);
}

std::vector<geometry_msgs::Twist>
RobotariumConsensus::saturateVelocity(std::vector<geometry_msgs::Twist>
dxi, double maxLinear, double maxAngular){

```

```

// Initialization
std::vector<geometry_msgs::Twist> dxi_out = dxi;

for (int i = 0; i < dxi.size(); ++i){
    // Saturate linear velocity
    if (dxi[i].linear.x > maxLinear) dxi_out[i].linear.x =
maxLinear;
    if (dxi[i].linear.x < -maxLinear) dxi_out[i].linear.x = -
maxLinear;

    // Saturate angular velocity
    if (dxi[i].angular.z > maxAngular) dxi_out[i].angular.z =
maxAngular;
    if (dxi[i].angular.z < -maxAngular) dxi_out[i].angular.z = -
maxAngular;
}

return dxi_out;
}

int RobotariumConsensus::nChooseK(int iN, int iR){
    if (iR < 0 || iR > iN) {
        return 0;
    }

    // Initialization
    int iComb = 1;
    int i = 0;

    // Iterative loop
    while (i < iR) {
        ++i;
        iComb *= iN - i + 1;
        iComb /= i;
    }

    return iComb;
}

std::vector<int> RobotariumConsensus::topologicalNeighbors(int i, int
N){
    std::vector<int> neighbors;
    // Special cases
    if (N == 1) return neighbors;
    if (N == 2){
        if (0 == i){
            neighbors.push_back(1);
        } else {
            neighbors.push_back(0);
        }
        return neighbors;
    }

    // Normal cases
    if (0 == i){
        neighbors.push_back(1);
        neighbors.push_back(N-1);
    }
}

```

```

} else if (N-1 == i){
    neighbors.push_back(0);
    neighbors.push_back(N-2);
} else {
    neighbors.push_back(i-1);
    neighbors.push_back(i+1);
}
return neighbors;
}

std::vector<geometry_msgs::Twist>
RobotariumConsensus::barrierCertificate(std::vector<geometry_msgs::Twis
t> dxi, const gazebo_msgs::ModelState::ConstPtr& msg, double gamma,
double safetyRadius){
    // Initialization
    int N = msg->pose.size();
    std::vector<geometry_msgs::Twist> dxi_out = dxi;

    // If only one robot, return same command immediately
    if (N < 2) return dxi_out;
    // Else

    // QP (1/2*x'*D*x + c'*x) with A*x <= b, unbounded variables
    Program qp (CGAL::SMALLER, false, 0, false, 0);

    // Initialize problem arrays
    int numConstraints = nChooseK(N,2);
    int count = 0;
    double h;
    double d,dx,dy,d2;

    // Set the non-default entries
    for (int i = 0; i < N-1; ++i){
        for (int j = i+1; j < N; ++j){
            // Compute norm
            dx = msg->pose[i].position.x - msg->pose[j].position.x;
            dy = msg->pose[i].position.y - msg->pose[j].position.y;
            d2 = dx*dx + dy*dy;
            h = d2 - safetyRadius*safetyRadius;

            qp.set_a(2*i,    count, -2.*dx);
            qp.set_a(2*i+1, count, -2.*dy);
            qp.set_a(2*j,    count, 2.*dx);
            qp.set_a(2*j+1, count, 2.*dy);

            qp.set_b(count, gamma*h*h*h);

            // Increment
            ++count;
        }
    }

    // Quadratic function
    for (int i = 0; i < 2*N; ++i) qp.set_d(i, i, 2.);

    for (int i = 0; i < N; ++i){
        qp.set_c(2*i, -2*dxi[i].linear.x);
    }
}

```

```

        qp.set_c(2*i+1, -2*dxi[i].linear.y);
    }

    // Solve the QP problem, using ET as the exact type
    Solution s = CGAL::solve_quadratic_program(qp, ET());

    ET denom = s.variables_common_denominator();
    unsigned int i = 0;
    for (Solution::Variable_numerator_iterator it =
s.variable_numerators_begin(); it < s.variable_numerators_end(); ++it,
++i){
        // X component
        dxi_out[i].linear.x =
CGAL::to_double(*it)/CGAL::to_double(denom);
        // Y component
        ++it;
        dxi_out[i].linear.y =
CGAL::to_double(*it)/CGAL::to_double(denom);
    }

    for (int i = 0; i < dxi_out.size(); ++i){
        std::cout << "Robot " << i << std::endl;
        std::cout << "\tIN = [" << dxi[i].linear.x << ", " <<
dxi[i].linear.y << "]" << std::endl;
        std::cout << "\tOUT = [" << dxi_out[i].linear.x << ", " <<
dxi_out[i].linear.y << "]" << std::endl;
    }

    return dxi_out;
}

gazebo_msgs::ModelState::ConstPtr
RobotariumConsensus::uniToSiStates(const
gazebo_msgs::ModelState::ConstPtr& msg, double projectionDistance){
    // Initialization
    int n = msg->pose.size();
    double roll, pitch, yaw;
    gazebo_msgs::ModelState msg_si;

    for (int i = 0; i < n; ++i){
        // Compute yaw
        tf::Quaternion q(msg->pose[i].orientation.x, msg-
>pose[i].orientation.y, msg->pose[i].orientation.z, msg-
>pose[i].orientation.w);
        tf::Matrix3x3 m(q);
        m.getRPY(roll, pitch, yaw);

        // Compute and assign single-integrator state
        geometry_msgs::Point point;
        point.x = msg->pose[i].position.x +
projectionDistance*cos(yaw);
        point.y = msg->pose[i].position.y +
projectionDistance*sin(yaw);
        point.z = 0.;
        geometry_msgs::Quaternion quat;
        quat.x = 0.;
        quat.y = 0.;
    }
}

```

```

    quat.z = 0.;
    quat.w = 1.;
    geometry_msgs::Pose pose;
    pose.position = point;
    pose.orientation = quat;

    msg_si.pose.push_back(pose);
    msg_si.twist.push_back(geometry_msgs::Twist());
}

// Return pointer
gazebo_msgs::ModelState::ConstPtr msg_si_ptr( new
gazebo_msgs::ModelState( msg_si ) );

return msg_si_ptr;
}

std::vector<geometry_msgs::Twist>
RobotariumConsensus::int2uni3(std::vector<geometry_msgs::Twist> dxi,
const gazebo_msgs::ModelState::ConstPtr& msg, double lambda){
    // Initialization
    double roll, pitch, yaw;
    std::vector<geometry_msgs::Twist> dxu;

    // Get yaw from quaternions
    for (int i = 0; i < msg->pose.size(); ++i){
        geometry_msgs::Twist vel;
        tf::Quaternion q(msg->pose[i].orientation.x, msg-
>pose[i].orientation.y, msg->pose[i].orientation.z, msg-
>pose[i].orientation.w);
        tf::Matrix3x3 m(q);
        m.getRPY(roll, pitch, yaw);

        // From single-integrator dynamics to unicycle dynamics
        vel.linear.x = cos(yaw)*dxi[i].linear.x +
sin(yaw)*dxi[i].linear.y;
        vel.angular.z = (1.0/lambda)*(-sin(yaw)*dxi[i].linear.x +
cos(yaw)*dxi[i].linear.y);

        //if (vel_out.linear.x < 0.0) vel_out.angular.z = - 1.0 *
vel_out.angular.z;

        dxu.push_back(vel);
    }

    return dxu;
}

double RobotariumConsensus::computeAverageDistanceChange(const
gazebo_msgs::ModelState::ConstPtr& msg){
    // Initialization
    int n = msg->pose.size();
    double d = 0;
    double dx = 0;
    double dy = 0;

    // Compute average distance change

```



```

    for (int i = 0; i < n; ++i){
        // Compute coordinate changes
        dx = msg->pose[i].position.x - previous_x[i];
        dy = msg->pose[i].position.y - previous_y[i];

        // Add distance change to sum
        d += sqrt(dx*dx + dy*dy);

        // Update
        previous_x[i] = msg->pose[i].position.x;
        previous_y[i] = msg->pose[i].position.y;
    }

    // Average
    return d/n;
}

geometry_msgs::Twist RobotariumConsensus::computeRobotVelocity(int
robotID, const geometry_msgs::Pose pose, const
gazebo_msgs::ModelState::ConstPtr& msg, double projectionDistance){
    // Initialization
    geometry_msgs::Twist vel;
    int n = msg->pose.size();

    // Get neighbors of agent
    std::vector<int> neighbors = topologicalNeighbors(robotID, n);

    // Compute velocity
    for (std::vector<int>::iterator it = neighbors.begin(); it !=
neighbors.end(); ++it){
        vel.linear.x += msg->pose[*it].position.x - msg-
>pose[robotID].position.x;
        vel.linear.y += msg->pose[*it].position.y - msg-
>pose[robotID].position.y;
    }

    return vel;
}

void RobotariumConsensus::poseCallback(const
gazebo_msgs::ModelState::ConstPtr& msg)
{
    // Initialization
    geometry_msgs::Twist vel;
    int n = msg->pose.size();
    double d = 0;

    if (this->init){
        // For each robot
        for (int i = 0; i < n; ++i){
            // Create topic name
            std::ostringstream topic_name;
            topic_name << "/" << msg->name[i] << "/cmd_vel";

            // Initialize publishing topic

```

```

myvector.push_back(nh_.advertise<geometry_msgs::Twist>(topic_name.str()
, 10));

    // Initialize previous_msg
    previous_x.push_back(msg->pose[i].position.x);
    previous_y.push_back(msg->pose[i].position.y);
}
this->init = false;
} else {
    // If stopping condition is reached, stop simulation
    // This node should be instantiated as required so that if it
stops, all other nodes stop
    double d = computeAverageDistanceChange(msg);
    std::cout << "[t = " << ros::Time::now() << "] " << d <<
std::endl;
    if (d < 0.00001){
        std::cout << "\tCounting " << count << std::endl;
        ++count;
    } else {
        count = 0;
    }
    // After N measures below the threshold, stop
    // After 2.5 minutes stop
    // ros::Time::now().toSec() > 1.2/this->v
    if (count > 25 || ros::Time::now().toSec() > 150.0){
        // Stop node
        ros::shutdown();
    }
}

// Convert to single-integrator states
double projectionDistance = 0.03;
gazebo_msgs::ModelState::ConstPtr msg_si = uniToSiStates(msg,
projectionDistance);
std::vector<geometry_msgs::Twist> dxi, dxu;

// For each robot
for (int i = 0; i < n; ++i){
    // Compute velocity
    dxi.push_back(computeRobotVelocity(i, msg->pose[i], msg_si,
projectionDistance));
}

// Compute collision-free controls
dxi = barrierCertificate(dxi, msg_si, 10000., .06);

// Transform single-integrator dynamics to unicycle dynamics
dxu = int2uni3(dxi, msg, projectionDistance);

// Saturation
dxu = saturateVelocity(dxu, this->v, 2.0*3.1416);

// Publish to each robot input topic
for (int i = 0; i < n; ++i){
    // Model identification
    dxu[i].linear.x *= .835;
}

```

```

        dxu[i].angular.z *= .46;

        myvector[i].publish(dxu[i]);
    }

    // Re-publish poses
    poses_pub.publish(msg);
}

void RobotariumConsensus::setMaxLinearVelocity(double
maxLinearVelocity)
{
    this->v = maxLinearVelocity;
}

int main(int argc, char** argv)
{
    // Initialization
    ros::init(argc, argv, "consensus");
    RobotariumConsensus consensus;

    if (argc == 2){
        consensus.setMaxLinearVelocity(atof(argv[1]));
    }

    // Wait for some time for Gazebo to be ready
    ros::Duration(5.0).sleep();

    // Iterate at given rate
    ros::Rate rate(30.);
    while(ros::ok())
    {
        // Activate callbacks
        ros::spinOnce();
        // Maintain publishing rate
        rate.sleep();
    }

    return 0;
}

```

C.4.2 Tracker

Always running as a backup system to the computer vision node calculating the poses of the ID tags of the robots, the *tracker* node bypasses this latter by reading the poses of the robots directly from the Gazebo simulator. In return, it outputs these poses under the correct format in the appropriate ROS topic.

tracker.cpp

```

#include <ros/ros.h>
#include <gazebo_msgs/ModelStates.h>

class RobotariumTracker
{
public:
    // Constructor
    RobotariumTracker();

private:
    // Attributes
    ros::NodeHandle nh_;
    ros::Subscriber camera_sub;
    ros::Publisher poses_pub;
    gazebo_msgs::ModelStates poses;

    // Methods
    void cameraCallback(const gazebo_msgs::ModelStates::ConstPtr& msg);

public:
    void publish();
};

RobotariumTracker::RobotariumTracker()
{
    // Subscribe to the tracking camera topic
    camera_sub = nh_.subscribe("/gazebo/model_states", 1,
&RobotariumTracker::cameraCallback, this);
    // Create topic to publish robots poses
    poses_pub =
nh_.advertise<gazebo_msgs::ModelStates>("/robotarium/poses", 1);
}

void RobotariumTracker::cameraCallback(const
gazebo_msgs::ModelStates::ConstPtr& msg)
{
    // Re-initialize poses
    poses.name.clear();
    poses.pose.clear();

    // Get robots poses #####
    // TODO: replace by the tracking the AprilTags
    for (int i = 0; i < msg->pose.size(); ++i){
        if ("gritbot" == msg->name[i].substr(0,7)){
            poses.name.push_back(msg->name[i]);
            poses.pose.push_back(msg->pose[i]);
        }
    }
    // #####
}

void RobotariumTracker::publish(){
    // Publish
    poses_pub.publish(poses);
}

int main(int argc, char** argv)

```

```

{
  ros::init(argc, argv, "tracker");
  RobotariumTracker tracker;

  // Iterate at given rate
  ros::Rate rate(30.);
  while(ros::ok())
  {
    // Activate callbacks
    ros::spinOnce();
    // Publish tracked poses
    tracker.publish();
    // Maintain publishing rate
    rate.sleep();
  }

  return 0;
}

```

C.4.3 Logger

Essential part of the navigation package of the microscopic model, the *logger* node saves the simulation time as well as the successive poses of each robot in a file during the consensus mission. This file can later be analyzed to retrieve trajectories and compute the main mission metrics such as consensus location and consensus time.

logger.cpp

```

#include <gazebo_msgs/ModelStates.h>
#include <fstream>
#include <sstream> // for ostringstream
#include <ros/ros.h>

using namespace std;

class RobotariumLogger
{
public:
  // Constructor
  RobotariumLogger();
  void setFileName(const char *filename);

private:
  // Attributes
  ofstream myfile;
  ros::NodeHandle nh_;
  ros::Subscriber poses_sub_;

  // Methods

```

```

    void poseCallback(const gazebo_msgs::ModelState::ConstPtr& msg);
};

RobotariumLogger::RobotariumLogger()
{
    // Declare topics
    poses_sub_ = nh_.subscribe("/robotarium/controller/poses", 1,
&RobotariumLogger::poseCallback, this);
}

void RobotariumLogger::setFileName(const char *filename){
    // Open logging file
    myfile.open(filename);
}

void RobotariumLogger::poseCallback(const
gazebo_msgs::ModelState::ConstPtr& msg)
{
    // Initialization
    ros::Time time = ros::Time::now();
    int n = msg->pose.size();
    ostringstream line;
    line << time << ",";

    // For each robot
    for (int i = 0; i < n; ++i){
        // Concatenate poses information
        line << msg->pose[i].position.x;
        line << ",";
        line << msg->pose[i].position.y;
        if (i < n-1) line << ",";
    }

    // Add poses to file
    myfile << line.str() << "\n";
}

int main(int argc, char** argv)
{
    ros::init(argc, argv, "logger");
    RobotariumLogger logger;

    // Set filename
    if (argc == 2){
        logger.setFileName(argv[1]);
    } else {
        logger.setFileName("log.csv");
    }

    // Iterate at given rate
    ros::Rate rate(30.);
    while(ros::ok())
    {
        // Activate callbacks
        ros::spinOnce();
        // Maintain publishing rate
        rate.sleep();
    }
}

```

```
    }  
    // TODO: ideally there should be a node shutdown callback to close  
the file stream  
    //myfile.close();  
  
    return 0;  
}
```

REFERENCES

- [1] P. Chatterjee, "The Three Faces of Drone War: Speaking Truth From the Robotic Heavens," *The UNZ Review: An Alternative Media Selection*, 2014.
- [2] S. Genouel, *Systèmes linéaires continus et invariants*, 2011.
- [3] J. de Hoog, S. Cameron and A. Visser, "Role-Based Autonomous Multi-robot Exploration," in *Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns, 2009. COMPUTATIONWORLD '09. Computation World.*, 2009.
- [4] S. Schröder, *Optimized Movements: Ballet of the Bots*, 2014.
- [5] Siemens, *Smoothly moving industrial robots save energy*, 2014.
- [6] Aeroweb, *U.S. Military Aircraft Programs*, <http://www.bga-aeroweb.com/DoD-Aircraft-Programs.html>, 2015.
- [7] O. of the Under Secretary of Defense (Comptroller) Chief Financial Officer, "Program Acquisition Cost by Weapon System," http://comptroller.defense.gov/Portals/45/Documents/defbudget/fy2015/fy2015_Weapons.pdf, 2014.
- [8] W. Wheeler, *How Much Does an F-35 Actually Cost?*, 2014.
- [9] Deagel.com, *C-40A Clipper*, 2015.
- [10] A. Navy, *United States Navy Fact File: C-2A Greyhound Logistics Aircraft*, 2013.
- [11] Military.com, *EP-3 Ares II*, 2015.
- [12] A. Navy, *United States Navy Fact File: C-9 Skytrain Logistics Aircraft*, 2009.

- [13] G. Aircraft, *AV-8B Harrier II*, 2013.
- [14] M. A. Network, *C-20*, 1998.
- [15] U. A. Force, *A-10 Thunderbolt II*,
<http://www.af.mil/AboutUs/FactSheets/Display/tabid/224/Article/104490/a-10-thunderbolt-ii.aspx>, 2015.
- [16] A. C. Command, *MC-12W Liberty Project Aircraft (LPA)*, 2011.
- [17] R. Goyer, *Pilatus PC-123 Versus the World*, 2013.
- [18] U. A. Force, *T-1A Jayhawk*,
<http://www.af.mil/AboutUs/FactSheets/Display/tabid/224/Article/104542/t-1a-jayhawk.aspx>, 2005.
- [19] L. S. J. E. R. Johnson, *American Military Training Aircraft: Fixed and Rotary-Wing Trainers Since 1916*, McFarland, 2015.
- [20] AOPA, *Quick Look: Learjet 35/36*, <http://www.aopa.org/News-and-Video/All-News/2013/May/1/Quick-Look-Learjet-35-36>, 2013.
- [21] J. Pike, *Bell 206 JetRanger*, 2015.
- [22] E. Gent, *Price wars: Counting the cost of drones, planes and satellites*, 2015.
- [23] DJI, *Phantom 3 Professional*, <https://www.dji.com/product/phantom-3-pro>, 2015.
- [24] H.-L. G. Solutions, *Sensefly Ebee RTK*, <http://www.hlgs.com.au/uavs/sensefly-ebee-rtk/>, 2015.
- [25] T. Dustrude, *Cessna at Westwind*, <http://sanjuanupdate.com/2013/07/34100/>, 2013.

- [26] NASA, *Landsat 8 Status Update for Aug. 8, 2013*,
<http://landsat.gsfc.nasa.gov/?p=6099>, 2013.
- [27] M. Thompson, "Costly Flight Hours," *Time*, April 2013.
- [28] A. Yates, *Pricing*, <http://www.kerrvillephoto.com/index.php/aerial-pricing>, 2015.
- [29] D. W. Felty, *Aerial Photography Fees & Services*, 2006.
- [30] J. Pinckert, *Drone vs. Helicopter for Aerials: Top 5 Scenarios for Each*, 2014.
- [31] A. Matese, P. Toscano, S. F. Di Gennaro, L. Genesio, F. P. Vaccari, J. Primicerio, C. Belli, A. Zaldei, R. Bianconi and B. Gioli, "Intercomparison of UAV, Aircraft and Satellite Remote Sensing Platforms for Precision Viticulture," *Remote Sensing*, vol. 7, no. 3, p. 2971, 2015.
- [32] C. Mailey, "Are UAS More Cost Effective than Manned Flights?," 2013.
- [33] Lucintel, "Growth Opportunity in Global UAV Market," 2011.
- [34] marketsandmarkets.com, "Unmanned Aerial Vehicles Market by Class (Small, Tactical, Strategic, Special Purpose), Subsystem (Data Link, GCS, and Software), Application (Military, Commercial and Homeland Security), Procurement by Purpose (Procurements, RDT&E, O&M), Payload & Geography - Global Forecast to 2020," 2015.
- [35] I. IGI Consulting, "UAV Market Research Study – 2014 Edition," 2014.
- [36] M. Ballve, *The Drone Report: Market Forecast For Commercial Applications, Regulatory Process, And Leading Players*, 2015.
- [37] C. Snow, *Diversity and Hype in Commercial Drone Market Forecasts*, 2015.

- [38] M. McFarland, *Amazon details its plan for how drones can fly safely over U.S. skies*, 2015.
- [39] M. Bedford, "Unmanned Aircraft System (UAS) Service Demand 2015-2035: Literature Review & Projections of Future Usage," 2013.
- [40] M. Lukovic, *The Future of the Civil and Military UAV Market*, 2011.
- [41] Amazon.com, *Amazon Prime Air*, 2015.
- [42] Google, *Introducing Project Wing*, 2014.
- [43] Bionic, *Aquila Facebook Drone / Internet.org*, 2015.
- [44] C. Anderson, "Agricultural Drones: Relatively cheap drones with advanced sensors and imaging capabilities are giving farmers new ways to increase yields and reduce crop damage.," 2014.
- [45] J. Reagan, "How Low Will Drone Prices Go?," *DroneLife.com*, July 2015.
- [46] P. Belton, "Game of drones: As prices plummet drones are taking off," *BBC News*, January 2015.
- [47] PriceSpy, *Parrot AR. Drone 2.0 Elite Edition RTF Price History*, 2015.
- [48] S. Curtis, "Parrot launches 'camouflaged' AR Drone," *The Telegraph*, December 2013.
- [49] DJI, *Price Reduction for the Phantom 2 Series Drones*, 2014.
- [50] Amazon.com, *Amazon.com: Online Shopping for Electronics, Apparel, Computers, Books, DVDs and more*, <http://www.amazon.com/>, 2015.

- [51] W. P. Library, *List of Unmanned Aerial Vehicles*, 2015.
- [52] D. B. German, "The Need for Numerical Optimization," 2013.
- [53] K. Smith, *Best Drone For Sale and Why*, 2015.
- [54] DJI, *Flame Wheel ARF Kit*, 2016.
- [55] 2. C. A. A. Race, *The Drone Index: IAI Super Heron*, 2015.
- [56] M. Deveau, *Image recording from aerial vectors*, 2009.
- [57] D. Engineering, *VTOL Vehicle Design and Development*, 2013.
- [58] K. Barnstorff, *NASA engineers have successfully built and flown a 10-engine electric-hybrid aerial UAV beast*, 2015.
- [59] D. L. Gareth Roberts and W. Crowther, "Non-Planar Hexrotor: Concept & Missions," 2011.
- [60] I. I. S. de l'Aéronautique et de l'Espace, *Des innovations ISAE aux débouchés commerciaux*, 2013.
- [61] Range, *SEED Concept: Your Personal Automonous Floating Camera Drone*, 2011.
- [62] T. Digest, *CES 2008: Da Vinci inspired remote control miniatures*, 2008.
- [63] R. Beckler, "This Awesome Drone Can Conquer Both Sea And Sky," *Vocativ*, December 2015.
- [64] S. H. Dmitry Bershinsky and E. N. Johnson, *Underwater Quadrotor Flight Test*, 2015.
- [65] C. Anderson, *DYIDrones.com*, 2007.

- [66] ArduPilot.com, *ArduPilot Autopilot Suite*, <http://ardupilot.com/>, 2015.
- [67] D. J. Pate, M. D. Patterson and B. J. German, "Optimizing families of reconfigurable aircraft for multiple missions," *Journal of Aircraft*, vol. 49, no. 6, pp. 1988-2000, 2012.
- [68] R. Hoover, "It's Alive! Ames Engineers Harvest and Print Parts for New Breed of Aircraft," *NASA*, October 2015.
- [69] D. L. K. D. C. D. N. M. Zachary C. Fisher, "ADAPt Design: A Methodology for Enabling Modular Design for Mission Specific SUAS," 2016.
- [70] C. L. R. E. S. K. D. C. D. N. M. David B. Locascio, "A Framework for Integrated Analysis, Design, and Rapid Prototyping of Small Unmanned Airplanes," 2016.
- [71] P. Mangum, Z. Fisher, K. D. Cooksey, D. Mavris, E. Spero and J. W. Gerdes, "An Automated Approach to the Design of Small Aerial Systems Using Rapid Manufacturing," in *ASME 2015 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, 2015.
- [72] L. Zyga, *Panasonic's Si-alloy anode technology to offer 30% increase in battery capacity*, 2010.
- [73] G. Gottsegen, "The Parrot Bebop 2 Drone Has Doubled Its Battery Life," *Wired*, November 2015.
- [74] Parrot, *Technical Specifications of AR.Drone 1.0*, http://www.ardrone-flyers.com/wiki/Technical_Specifications_of_AR.Drone_1.0, 2010.

- [75] M. Cooney, *Drones still face major communications challenges getting onto US airspace*, 2013.
- [76] K. Moskvitch, "Are drones the next target for hackers?," *BBC.com*, February 2014.
- [77] C. Howard, "UAV command, control & communications," *Military & Aerospace*, July 2013.
- [78] Parrot, *Wifi Use: How to respect regulations and optimize Bebop Drone range*, 2014.
- [79] W. Saletan, "Fukushima's Bio-Robots: In Japan's nuclear cleanup, is human life cheaper than machines?," *Slate*, April 2011.
- [80] N. Michael, S. Shen, K. Mohta, Y. Mulgaonkar, V. Kumar, K. Nagatani, Y. Okada, S. Kiribayashi, K. Otake, K. Yoshida, K. Ohno, E. Takeuchi and S. Tadokoro, "Collaborative mapping of an earthquake-damaged building via ground and aerial robots," *Journal of Field Robotics*, vol. 29, no. 5, pp. 832-841, 2012.
- [81] L. E. Parker, "Heterogeneous multi-robot cooperation," 1994.
- [82] DoD, "Unmanned Systems Integrated Roadmap," 2013.
- [83] R. Pater, "Drone Survival Guide: Twenty-first Century Birdwatching," <http://dronesurvivalguide.org/>, 2015.
- [84] F. Icon, *Free Vector Icons*, 2017.
- [85] R. Valdes, "How the Predator UAV Works," *Science: How Stuff Works*, November 2002.

- [86] J. Hsu, "Drone War Pushes Pilots to Breaking Point," *Discover: Science for the curious*, January 2015.
- [87] MCruz, *Person icon black*, 2014.
- [88] Avjobs, *Air Traffic Control*.
- [89] A. Aerospace, *Unmanned Aerial Vehicle*, 3rd East Steet, Kamaraj Nagar, Thiruvanmiyur, Chennai - 600041, 2015.
- [90] S. R. Service, *Commercial Roofing*, 2013.
- [91] E. Swanson, *Drone Poll Finds Support For Strikes, With Limits*, 2013.
- [92] A. I. Association, "Unmanned Aircraft Systems: Perceptions and Potential," 2013.
- [93] B. Schneier, *Is it OK to shoot down a drone over your backyard?*, 2015.
- [94] F. Mondada, "Ensembles and mobile robots, where is the link?," 2011.
- [95] M. Brambilla, E. Ferrante, M. Birattari and M. Dorigo, "Swarm robotics: a review from the swarm engineering perspective," *Swarm Intelligence*, vol. 7, no. 1, pp. 1-41, 2013.
- [96] I. Navarro and F. Matía, "An Introduction to Swarm Robotics," *ISRN Robotics*, vol. 2013, no. Article ID 608164, p. 10, 2013.
- [97] Favindex, *A ball of mackerel fish defends against sea predators*,
<http://favindex.com/image/102116582886>.
- [98] *Birds flock swarm vortex Animals*,
http://wallpaperbeta.com/thumbnail/birds_flock_swarm_vortex_animals_hd-wallpaper-393564.jpg.

- [99] D. Story, "Swarm Intelligence: An Interview with Eric Bonabeau," *Open P2P*, 2003.
- [100] T. Seeley, "Consensus building during nest-site selection in honey bee swarms: The expiration of dissent," *Behavioral Ecology and Sociobiology*, vol. 53, no. 6, pp. 417-424, 2003.
- [101] G. Dudek, M. Jenkin, E. Milios and D. Wilkes, "A taxonomy for multi-agent robotics," *Autonomous Robots*, vol. 3, no. 4, pp. 375-397, 1996.
- [102] L. Iocchi, D. Nardi and M. Salerno, "Reactivity and deliberation: a survey on multi-robot systems," in *Balancing reactivity and social deliberation in multi-agent systems*, Springer, 2001, pp. 9-32.
- [103] E. Sahin, "Swarm robotics: From sources of inspiration to domains of application," in *Swarm robotics*, Springer, 2005, pp. 10-20.
- [104] G. Beni, "From swarm intelligence to swarm robotics," in *Swarm robotics*, Springer, 2005, pp. 1-9.
- [105] M. Dorigo and E. Sahin, "Guest editorial," *Autonomous Robots*, vol. 17, no. 2, pp. 111-113, 2004.
- [106] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," in *ACM Siggraph Computer Graphics*, 1987.
- [107] F. R. Noreils, "Toward a robot architecture integrating cooperation between mobile robots: Application to indoor environment," *The International Journal of Robotics Research*, vol. 12, no. 1, pp. 79-98, 1993.

- [108] P. Caloud, W. Choi, J.-C. Latombe, C. Le Pape and M. Yim, "Indoor automation with many mobile robots," in *Intelligent Robots and Systems' 90. Towards a New Frontier of Applications', Proceedings. IROS'90. IEEE International Workshop on*, 1990.
- [109] K. O. A. M. Y. I. H. Asama and I. Endo, "Development of task assignment system using communication for multiple autonomous robots," *Journal of Robotics and Mechatronics*, vol. 4, no. 2, pp. 122-127, 1992.
- [110] J. Wang, "DRS operating primitives based on distributed mutual exclusion," in *Intelligent Robots and Systems' 93, IROS'93. Proceedings of the 1993 IEEE/RSJ International Conference on*, 1993.
- [111] R. C. Arkin, Behavior-based robotics, MIT press, 1998.
- [112] F. Mondada, L. M. Gambardella, D. Floreano and M. Dorigo, "The cooperation of swarm-bots: Physical interactions in collective robotics," *IEEE Robot. Automat. Mag*, p. 2005.
- [113] K. Sreenath, T. Lee and V. Kumar, "Geometric control and differential flatness of a quadrotor UAV with a cable-suspended load," in *Decision and Control (CDC), 2013 IEEE 52nd Annual Conference on*, 2013.
- [114] K. Sreenath and V. Kumar, "Dynamics, control and planning for cooperative manipulation of payloads suspended by cables from multiple quadrotor robots," *rn*, vol. 1, no. r2, p. r3, 2013.
- [115] D. Aksaray, "Formulation of Control Strategies for Requirement Definition of Multi-Agent Surveillance Systems," Georgia Institute of Technology, 2014.

- [116] V. Harrison, *Robots to build canal bridge using 3D printing*, 2015.
- [117] P. D. Mavris, "UV-CORE: Unmanned Vehicle Collaboration Research Environment," 2015.
- [118] J. D. Sutter, *How 9/11 inspired a new era of robotics*, 2011.
- [119] Aerovironment, *Search & Rescue*, 2016.
- [120] A. a. M. D. a. P. C. a. K. V. Kushleyev, "Towards a swarm of agile micro quadrotors," *Autonomous Robots*, vol. 35, no. 4, pp. 287-300, 2013.
- [121] D. Hambling, *U.S. Navy Plans to Fly First Drone Swarm This Summer*, 2016.
- [122] D. Hambling, *Drone swarms will change the face of modern warfare*, 2016.
- [123] S. S. Ltd, *Future Prospects*, 2016.
- [124] L. A. Amaral and J. M. Ottino, "Complex networks," *The European Physical Journal B-Condensed Matter and Complex Systems*, vol. 38, no. 2, pp. 147-162, 2004.
- [125] M. W. Maier, "Architecting principles for systems-of-systems," in *INCOSE International Symposium*, 1996.
- [126] O. Holland, J. Woods, R. De Nardi and A. Clarck, "Beyond swarm intelligence: the ultraswarm," 2005.
- [127] R. De Nardi and O. Holland, "Swarmav: A swarm of miniature aerial vehicles," 2006.
- [128] W. L. Teacy, J. Nie, S. McClean, G. Parr, S. Hailes, S. Julier, N. Trigoni and S. Cameron, "Collaborative sensing by unmanned aerial vehicles," 2009.

- [129] D. Smalley, "LOCUST: Autonomous, swarming UAVs fly into the future," *Office of Naval Research*, April 2015.
- [130] A. Ollero and I. Maza, *Multiple Heterogeneous Unmanned Aerial Vehicles*, 1st ed., Springer Publishing Company, Incorporated, 2007.
- [131] J. Anderson, *Aircraft performance and design*, WCB/McGraw-Hill, 1999.
- [132] E. Saad, J. Vian, G. Clark and S. Bieniawski, "Vehicle swarm rapid prototyping testbed," in *AIAA Infotech@ Aerospace Conference and AIAA Unmanned... Unlimited Conference*, 2009.
- [133] L. Parker, "An Experiment in Mobile Robotic Cooperation," in *In Proceedings of the ASCE Specialty Conference on Robotics for Challenging Environments*, 1993.
- [134] M. J. Mataric, "Issues and approaches in the design of collective autonomous agents," *Robotics and Autonomous Systems*, vol. 16, no. 4, pp. 321-331, 1995.
- [135] M. Dorigo, D. Floreano, L. M. Gambardella, F. Mondada, S. Nolfi, T. Baaboura, M. Birattari, M. Bonani, M. Brambilla, A. Brutschy and others, "Swarmanoid: a novel concept for the study of heterogeneous robotic swarms," *Robotics & Automation Magazine, IEEE*, vol. 20, no. 4, pp. 60-71, 2013.
- [136] F. Ducatelle, G. A. Di Caro and L. M. Gambardella, "Cooperative Self-organization in a Heterogeneous Swarm Robotic System," in *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*, New York, NY, USA, 2010.

- [137] R. Grabowski, L. E. Navarro-Serment, C. J. Paredis and P. K. Khosla, "Heterogeneous teams of modular robots for mapping and exploration," *Autonomous Robots*, vol. 8, no. 3, pp. 293-308, 2000.
- [138] A. Howard, L. E. Parker and G. S. Sukhatme, "Experiments with a large heterogeneous mobile robot team: Exploration, mapping, deployment and detection," *The International Journal of Robotics Research*, vol. 25, no. 5-6, pp. 431-447, 2006.
- [139] A. Billard, A. J. Ijspeert and A. Martinoli, "A multi-robot system for adaptive exploration of a fast-changing environment: Probabilistic modeling and experimental study," *Connection Science*, vol. 11, no. 3-4, pp. 359-379, 1999.
- [140] V. Kim, "A Design Space Exploration Method for Identifying Emergent Behavior in Complex Systems," 2014.
- [141] D. N. Mavris and D. A. DeLaurentis, "An integrated approach to military aircraft selection and concept evaluation," 1995.
- [142] S. T. Kazadi, "Swarm engineering," 2000.
- [143] J. Kramer and M. Scheutz, "Development environments for autonomous mobile robots: A survey," *Autonomous Robots*, vol. 22, no. 2, pp. 101-132, 2007.
- [144] H. Kwong and C. Jacob, "Evolutionary exploration of dynamic swarm behaviour," *The 2003 Congress on Evolutionary Computation, CEC '03.*, vol. 1, December 2003.

- [145] H. Hamann and H. Worn, "A framework of space-time continuous models for algorithm design in swarm robotics," *Swarm Intelligence*, vol. 2, no. 2-4, pp. 209-239, 2008.
- [146] A. Burkle and S. Leuchter, "Development of Micro UAV Swarms," in *Autonome Mobile Systeme 2009*, Springer, 2009, pp. 217-224.
- [147] J. L. a. P. J. a. P. P. a. C. A. a. C. P. Sanchez-Lopez, "ROBOT2013: First Iberian Robotics Conference: Advances in Robotics, Vol.2," A. M. a. S. A. a. F. M. Armada, Ed., Cham, Springer International Publishing, 2014, pp. 55-63.
- [148] J. T. Allison, "Complex system optimization: A review of analytical target cascading, collaborative optimization, and other formulations," 2004.
- [149] N. P. Tedford and J. R. Martins, "Benchmarking multidisciplinary design optimization algorithms," *Optimization and Engineering*, vol. 11, no. 1, pp. 159-183, 2010.
- [150] N. M. Alexandrov, M. Y. Hussaini and others, *Multidisciplinary design optimization: state of the art*, vol. 80, SIAM, 1997.
- [151] L. U. Hansen and P. Horst, "Multilevel optimization in aircraft structural design evaluation," *Computers & structures*, vol. 86, no. 1, pp. 104-118, 2008.
- [152] F. Burgaud, C. Frank and D. N. Mavris, "A Business-Driven Optimization Methodology Applied to Suborbital Vehicle Programs," in *AIAA SPACE 2016*, 2016, p. 5248.

- [153] P. K. David Simchi-Levi and E. Simchi-Levi, *Designing & Managing the Supply Chain*, McGraw-Hill, 2003.
- [154] P. Yang, H. Wee, S. Chung and P. Ho, "Sequential and global optimization for a closed-loop deteriorating inventory supply chain," *Mathematical and Computer Modelling*, vol. 52, no. 12, pp. 161-176, 2010.
- [155] S. Mirjalili and A. Lewis, "Novel performance metrics for robust multi-objective optimization algorithms," *Swarm and Evolutionary Computation*, vol. 21, pp. 1-23, 2015.
- [156] R. L. Rardin and R. Uzsoy, "Experimental evaluation of heuristic optimization algorithms: A tutorial," *Journal of Heuristics*, vol. 7, no. 3, pp. 261-304, 2001.
- [157] C. Acquisition, A. Board, D. Sciences and N. Council, *Pre-Milestone A and Early-Phase Systems Engineering:: A Retrospective Review and Benefits for Future Air Force Acquisition*, National Academies Press, 2008.
- [158] D. K. Griendling, "Architecture-Based SoS Engineering: Introduction and Overview," 2013.
- [159] Wiley, *INCOSE Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities*, Wiley, 2015.
- [160] P. D. Mavris, *Architecture-Based SoS Engineering: Modeling Techniques Overview*, 2013.

- [161] O. Soysal and others, "Probabilistic aggregation strategies in swarm robotic systems," in *Swarm Intelligence Symposium, 2005. SIS 2005. Proceedings 2005 IEEE*, 2005.
- [162] M. Brambilla, C. Pinciroli, M. Birattari and M. Dorigo, "Property-driven design for swarm robotics," in *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, 2012.
- [163] L. E. Parker, "L-ALLIANCE: Task-oriented multi-robot learning in behavior-based systems," *Advanced Robotics*, vol. 11, no. 4, pp. 305-322, 1996.
- [164] R. Vaughan, "Massively multi-robot simulation in stage," *Swarm Intelligence*, vol. 2, no. 2, pp. 189-208, 2008.
- [165] C. Pinciroli, V. Trianni, R. O'Grady, G. Pini, A. Brutschy, M. Brambilla, N. Mathews, E. Ferrante, G. Di Caro, F. Ducatelle and others, "ARGoS: a modular, parallel, multi-engine simulator for multi-robot systems," *Swarm intelligence*, vol. 6, no. 4, pp. 271-295, 2012.
- [166] Y. Liu and K. M. Passino, "Stable social foraging swarms in a noisy environment," *Automatic Control, IEEE Transactions on*, vol. 49, no. 1, pp. 30-44, 2004.
- [167] I. Orlanski, "A rational subdivision of scales for atmospheric processes," *Bulletin of the American Meteorological Society*, vol. 56, pp. 527-530, 1975.
- [168] W. Burghout, "Mesoscopic simulation models for short-term prediction," *PREDIKT project report CTR2005*, vol. 3, 2005.

- [169] S. B. Van Hemel, J. MacMillan, G. L. Zacharias and others, *Behavioral Modeling and Simulation:: From Individuals to Societies*, National Academies Press, 2008.
- [170] G. E. Cantarella, S. De Luca, M. Di Gangi, R. Di Pace and S. Memoli, "Macroscopic vs. mesoscopic traffic flow models in signal setting design," in *Intelligent Transportation Systems (ITSC), 2014 IEEE 17th International Conference on*, 2014.
- [171] B. D. E. F. Nicolas Pujet, "Input-output modeling and control of the departure process of congested airports," *Guidance, Navigation, and Control Conference and Exhibit. Portland, OR, U.S.A.*, 1999.
- [172] W. Burghout, "Hybrid microscopic-mesoscopic traffic simulation," 2004.
- [173] C. P. Frank, O. J. Pinon-Fischer and D. N. Mavris, "A design space exploration methodology to support decisions under evolving requirements uncertainty and its application to suborbital vehicles," *53rd AIAA Aerospace Sciences Meeting, AIAA SciTech, (AIAA 2015-1010)*, 2015.
- [174] D. N. Mavris, D. S. Soban and M. C. Largent, "An Application of a Technology Impact Forecasting (TIF) Method to an Uninhabited Combat Aerial Vehicle," 1999.
- [175] M. R. Kirby, "A methodology for technology identification, evaluation, and selection in conceptual and preliminary aircraft design," 2001.
- [176] F. Villeneuve, "A method for concept and technology exploration of aerospace architectures," 2007.

- [177] M. Armstrong, "Function Based Architecture Design Space Definition and Exploration," *The 26th Congress of ICAS and 8th AIAA ATIO, Aviation Technology, Integration, and Operations (ATIO) Conferences*, 2008.
- [178] K. Sayler, "A World of Proliferated Drones," *Center for a New American Security*, 2015.
- [179] M. Michalko, *Thinkertoys: A Handbook of Creative-Thinking Techniques*, Potter/TenSpeed/Harmony, 2010.
- [180] K. Lewin, "Force field analysis," *The 1973 Annual Handbook for Group Facilitators*, pp. 111-13, 1946.
- [181] S. Savransky, *Engineering of Creativity: Introduction to TRIZ Methodology of Inventive Problem Solving*, CRC Press, 2000.
- [182] F. Zwicky, "New Methods of Thought and Procedure: Contributions to the Symposium on Methodologies," F. a. W. A. G. Zwicky, Ed., Berlin, Heidelberg, Springer Berlin Heidelberg, 1967, pp. 273-297.
- [183] S. R. Safavian and D. Landgrebe, "A survey of decision tree classifier methodology," 1990.
- [184] W. Engler, P. T. Biltgen and D. N. Mavris, "Concept selection using an interactive reconfigurable matrix of alternatives (IRMA)," in *45th AIAA Aerospace Sciences Meeting and Exhibit*, 2007.
- [185] Z. T. Mian, P. Dees, L. Hall and D. Mavris, "Development and Implementation of Micro Autonomous Systems and Technologies (MAST) Interactive Reconfigurable

- Matrix of Alternatives (M-IRMA) for Concept Selection," *Procedia Computer Science* , vol. 16, pp. 708-717, 2013.
- [186] C. Frank, J.-G. Durand, A. Levy, F. Allair and D. N. Mavris, "Design of an improved green taxiing system focused around the landing gear," *14th AIAA Aviation Technology, Integration, and Operations Conference, AIAA Aviation, (AIAA 2014-3010)*, 2014.
- [187] C. P. Frank, W. A. Levy, J.-G. D. Durand, E. Garcia and D. N. Mavris, "An Integrated and Parametric Environment for Generation, Selection and Evaluation of New Architectures at a Conceptual Level: Application to the Environmental Control System," *52nd Aerospace Sciences Meeting, AIAA SciTech, (AIAA 2014-0681)*, 2014.
- [188] C. F. Frederic Burgaud and D. N. Mavris, "An Aircraft Development Methodology Aligning Design and Strategy to Support Key Decision Making," *57th AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, AIAA SciTech*, January 2016.
- [189] V. K. Srivastava and A. Fahim, "An optimization method for solving mixed discrete-continuous programming problems," *Computers & Mathematics with Applications*, vol. 53, no. 10, pp. 1481-1491, 2007.
- [190] B. Kannan and S. N. Kramer, "An augmented Lagrange multiplier based method for mixed integer discrete continuous optimization and its applications to mechanical design," *Journal of mechanical design*, vol. 116, no. 2, pp. 405-411, 1994.

- [191] C. A. C. Coello, Evolutionary algorithms for solving multi-objective problems, vol. 242, Springer, 2002.
- [192] D. Wienke, C. Lucasius and G. Kateman, "Multicriteria target vector optimization of analytical procedures using a genetic algorithm: Part I. theory, numerical simulations and application to atomic emission spectroscopy," *Analytica Chimica Acta*, vol. 265, no. 2, pp. 211-225, 1992.
- [193] V. G. N., Multidiscipline Design Optimization, Vanderplaats Research & Development, 2007.
- [194] B. German, "Multi-Objective Optimization: Principles and Algorithms," 2013.
- [195] B. German, "Multidisciplinary Design Optimization (MDO): single-level and multi-level methods," 2013.
- [196] W. Yao, X. Chen, Q. Ouyang and Y. Wei, "A Concurrent Subspace Optimization Procedure Based on Multidisciplinary Active Regional Crossover Optimization," in *51st AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference 18th AIAA/ASME/AHS Adaptive Structures Conference 12th*, 2010.
- [197] G. Park, Analytic Methods for Design Practice, Springer London, 2007.
- [198] K. shi Zhang, "Aeronautics and Astronautics," M. Mulder, Ed., InTech, 2011.
- [199] S.-I. Yi, J.-K. Shin and G. Park, "Comparison of MDO methods with mathematical examples," *Structural and Multidisciplinary Optimization*, vol. 35, no. 5, pp. 391-402, 2008.

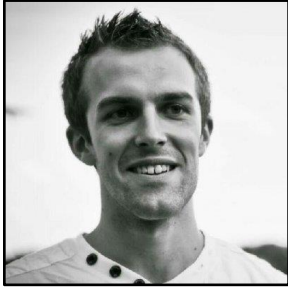
- [200] S. Kodiyalam and J. Sobieszczanski-Sobieski, "Bilevel integrated system synthesis with response surfaces," *AIAA journal*, vol. 38, no. 8, pp. 1479-1485, 2000.
- [201] A. Salamon, *Parallel Slowdown*, 2009.
- [202] J. L. Gustafson, "Reevaluating Amdahl's law," *Communications of the ACM*, vol. 31, no. 5, pp. 532-533, 1988.
- [203] F. P. Brooks, *The mythical man-month: Essays on Software Engineering*, vol. 1995, Addison-Wesley Reading, MA, 1975.
- [204] A. Mosteo, L. Montano and M. Lagoudakis, "Multi-robot routing under limited communication range," in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, 2008.
- [205] J. Melton, "Airships 101: Rediscovering the potential of lighter-than-air (LTA)," 2012.
- [206] E. C. C. for Accident and I. R. Systems, "ECCAIRS Aviation Data Definition Standard," 2013.
- [207] J. Meyer, A. Sendobry, S. Kohlbrecher, U. Klingauf and O. von Stryk, "Comprehensive simulation of quadrotor uavs using ros and gazebo," in *Simulation, Modeling, and Programming for Autonomous Robots*, Springer, 2012, pp. 400-411.
- [208] A. E. Ahmed, A. Hafez, A. Ouda, H. E. H. Ahmed and H. M. ABD-Elkader, "Modeling of a Small Unmanned Aerial Vehicle," *International Journal of Mechanical, Aerospace, Industrial, Mechatronic and Manufacturing Engineering*, vol. 9, no. 3, pp. 460-468, 2015.

- [209] S. Dufresne, C. Johnson and D. N. Mavris, "Variable fidelity conceptual design environment for revolutionary unmanned aerial vehicles," *Journal of Aircraft*, vol. 45, no. 4, pp. 1405-1418, 2008.
- [210] C. S. Chua, "Generic UAV modeling to obtain its aerodynamic and control derivatives," 2008.
- [211] C. Stachniss, "Exploration and mapping with mobile robots.," 2006.
- [212] J. A. S. Martins, MRSLAM-Multi-Robot Simultaneous Localization and Mapping, University of Coimbra, 2013.
- [213] E. Ferrante, "A control architecture for a heterogeneous swarm of robots: The design of a modular behavior-based architecture," 2009.
- [214] S. F. Railsback, S. L. Lytinen and S. K. Jackson, "Agent-based simulation platforms: Review and development recommendations," *Simulation*, vol. 82, no. 9, pp. 609-623, 2006.
- [215] G. T. U. R. Facility, *GUST Software*, 2016.
- [216] S. French, *How drones will drastically transform U.S. agriculture, in one chart*, 2015.
- [217] A. Press, *Agriculture the most promising market for drones*, 2013.
- [218] SkyPlan, *SkyPlan Services*, 2016.
- [219] senseFly, *Angry birds at altitude*, 2014.
- [220] W. Burgard, M. Moors, C. Stachniss and F. E. Schneider, "Coordinated multi-robot exploration," *Robotics, IEEE Transactions on*, vol. 21, no. 3, pp. 376-386, 2005.

- [221] S. Gumustekin, *An introduction to image mosaicing*, 1999.
- [222] P. Illsley, *GoPro Hero2 Aerial Imaging and Mapping Project*.
- [223] *Robotarium*, 2016.
- [224] D. G. P. W. L. M. M. A. A. F. E. & E. M. Pickem, "The Robotarium: A remotely accessible swarm robotics research testbed," *IEEE International Conference on Robotics and Automation (ICRA)*, September 2016.
- [225] D. Pickem, M. Lee and M. Egerstedt, "The GRITSBot in its natural habitat-a multi-robot testbed," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015.
- [226] D. Pickem, L. Wang, P. Glotfelter, Y. Diaz-Mercado, M. Mote, A. Ames, E. Feron and M. Egerstedt, "Safe, Remote-Access Swarm Robotics Research on the Robotarium," *arXiv preprint arXiv:1604.00640*, 2016.
- [227] J.-G. D. Durand, F. G. Burgaud, K. Cooksey and D. N. Mavris, "A Methodology to Evaluate Tradeoffs between Individual Architecture Development and Numerality to Achieve Group Performance in Robotics Swarms," *System (NAS)*, vol. 3, p. 4, 2016.
- [228] A. R. Pritchett, *Overview of Aircraft Models for Simulation - Modelling and Simulation of Dynamic Systems*, 2014.
- [229] W. Garage, *Willow Garage: PR2 Robot*, 2016.
- [230] ROS.org, *Package summary: ar_track_alvar*, 2016.

- [231] J.-Y. Tinevez, *matlab-tree: a MATLAB class to represent the tree data structure*, <https://github.com/tinevez/matlab-tree>, 2015.
- [232] T. O. Society, *How to Design & Build Ornithopters*, The Ornithopter Society 118 Callodine Avenue Buffalo NY 14226 USA, 2016.
- [233] H. Rübiger, *Development, theory and practice of large ornithopter models*, 2016.
- [234] R. Chelouah and P. Siarry, "A continuous genetic algorithm designed for the global optimization of multimodal functions," *Journal of Heuristics*, vol. 6, no. 2, pp. 191-213, 2000.
- [235] O. Magnussen, M. Ottestad and G. Hovland, "Multicopter Design Optimization and Validation," *Modeling, Identification and Control*, vol. 36, no. 2, p. 67, 2015.
- [236] S. S. & D. Bingham, *Virtual Library of Simulation Experiments: Test Functions and Datasets*, <https://www.sfu.ca/~ssurjano/index.html>, 2015.

VITA



Jean-Guillaume Dominique Sébastien Durand was born on January 25, 1990 in Fontaine-lès-Dijon, France. After obtaining his Baccalauréat with summa cum laude distinction in 2008, he pursued the traditional French classes préparatoires cursus for two years before taking competitive exams for the entrance to engineering schools. Jean-Guillaume was then accepted at the top French aerospace school program: the Supaero program of the Institut Supérieur de l'Aéronautique et de l'Espace (ISAE) from which he obtained its M.Sc. in 2013. In Supaero, he had the opportunity to make sensible contributions to the institute by being in charge of the Micro Air Vehicle Club (MAV club) with two of his classmates. The club successfully took part in several IMAV competitions. By also enrolling in a dual-degree program, Jean-Guillaume joined the Georgia Institute of Technology in 2012 where he obtained a M.Sc. in Aerospace Engineering in 2014. During these two years, he took part in the Airbus Fly Your Ideas challenge (2nd round qualification) and the Dassault Aerospace Challenge (Dassault Prize). Finally, after choosing to pursue his research at the Aerospace Systems Design Laboratory (ASDL) under Dr. Mavris, Jean-Guillaume obtained his Ph.D. in early 2017.

Jean-Guillaume's research interest lies mainly in unmanned aerial vehicles as they represent the intersection between aerospace engineering disciplines (aerodynamics, propulsion, structures, flight dynamics, design optimization) and robotics problematics (computer vision, structure from motion, mapping). Outside of the lab, Jean-Guillaume enjoys mountaineering and hiking the outdoors, learning guitar, and sports in general, especially basketball and volleyball.