University of Louisville
ThinkIR: The University of Louisville's Institutional Repository

Electronic Theses and Dissertations

12-2009

Optimal trajectory generation with DMOC versus NTG : application to an underwater glider and a JPL aerobot.

Weizhong Zhang
University of Louisville

Follow this and additional works at: https://ir.library.louisville.edu/etd

Recommended Citation
Zhang, Weizhong, "Optimal trajectory generation with DMOC versus NTG : application to an underwater glider and a JPL aerobot." (2009). *Electronic Theses and Dissertations.* Paper 1633.
https://doi.org/10.18297/etd/1633

This Doctoral Dissertation is brought to you for free and open access by ThinkIR: The University of Louisville's Institutional Repository. It has been accepted for inclusion in Electronic Theses and Dissertations by an authorized administrator of ThinkIR: The University of Louisville's Institutional Repository. This title appears here courtesy of the author, who has retained all other copyrights. For more information, please contact thinkir@louisville.edu.

OPTIMAL TRAJECTORY GENERATION WITH DMOC VERSUS NTG: APPLICATION TO AN UNDERWATER GLIDER AND A JPL AEROBOT

By

Weizhong Zhang
B.Sc. 2002, Electrical Engineering and Automation, Harbing Engineering University
M.Sc. 2005, Control Theory and Control Engineering, Shanghai Jiaotong University

A Dissertation
Submitted to the Faculty of the
Graduate School of the University of Louisville
in Partial Fulfillment of the Requirements
for the Degree of

Doctor of Philosophy

Department of Electrical and Computer Engineering
University of Louisville
Louisville, Kentucky

December 2009

OPTIMAL TRAJECTORY GENERATION WITH DMOC VERSUS NTG: APPLICATION TO AN UNDERWATER GLIDER AND A JPL AEROBOT

By

Weizhong Zhang
B.Sc. 2002, Electrical Engineering and Automation, Harbing Engineering University
M.Sc. 2005, Control Theory and Control Engineering, Shanghai Jiaotong University

A Dissertation Approved on

by the Following Reading and Examination Committee:

Tamer Inanc, Ph.D., Dissertation Director

Joseph D. Cole, Ph.D.

Ibrahim N. Imam, Ph.D.

Jerrold E. Marsden, Ph.D.

Jacek M. Zurada, Ph.D.

DEDICATION

This dissertation is dedicated to
my parents Zaiming Zhang and Xiaoya Qian, whose silent love and support
make me to be what I am today.

ACKNOWLEDGMENTS

My deepest gratitude goes to my advisor Dr. Tamer Inanc. I am a lucky student to have had the opportunity to meet Dr. Tamer Inanc. He provides me all he can to help me to grow as a Ph.D. student. Carefully reading my sometimes rash paper manuscripts, pointing out possible research directions, helping me to avoid unnecessary obstacles in research projects. All he did for me as an advisor will have a lasting impact on me wherever my career path takes me. The training and guidance I received in his directions will be also a valuable asset to assist me to solve different practical problems in the future.

I am so fortunate that Dr. Jerrold E. Marsden from Caltech could take the time to be one of my Ph.D. committee members. During my past projects, his insights and acuteness were shown to me in how research can be done with complete considerations including right directions to solve problems. I thank Dr. Sina Ober-Blöbaum from Caltech, Dr. Alberto Elfes from JPL for offering me the research collaboration experience which enhances my ability to solve problems. Wholeheartedly, I thank Dr. Jacek M. Zurada who was the Chair of the ECE department when I came to UofL. Without his recruiting me as a graduate student to U of L. I may have ended working as an engineer in Shanghai, and I may have chosen a totally different career path from what I am today. The growth and improvement in my ability and power are obvious during the Ph.D. study. I am so glad that I had this opportunity to come here. Gratitude goes to Dr. Joseph D. Cole and Dr. Ibrahim N. Imam for serving in my committee. I attended Dr. Cole's digital control class, and he is always ready to help me and give me some practical suggestions to enhance my knowledge and expertise in this specific field.

In conclusion, U of L as a fast-growing university in academics, is a very nice place in which to study and live. I personally have the chance to go to President Dr. James R. Ramsey's home to enjoy his family's courtesy. I thank Dr. Ramsey and Provost Shirley Willihnganz's leadership, all the faculty and staff members in our university are so nice and I am feeling welcome everywhere. Our department's Ph. D. Program Secretary Lisa Bell is always helping me whenever I have questions or problems. My friends at U of L, Sara, Elom, Travis, Dr. Dongqing Chen, Lijun Zhang, Qiang Ao, Liang Yang, Zhiyong Zhang, Gang Zhao, Yinan Cui and Hui Wang and so on helped me during the study. All others not mentioned here, thank you all for the help and kindness.

At last but not the least, I am thankful for the University of Louisville Fellowship, ECE teaching assistantship, and KY-NASA EPSCoR (contract WKURF 596855-08-02) for supporting me throughout my years of study at U of L.

ABSTRACT

OPTIMAL TRAJECTORY GENERATION WITH DMOC VERSUS NTG: APPLICATION TO AN UNDERWATER GLIDER AND A JPL AEROBOT

Weizhong Zhang

October 5, 2009

Optimal trajectory generation is an essential part for robotic explorers to execute the total exploration of deep oceans or outer space planets while curiosity of human and technology advancements of society both require robots to search for unknown territories efficiently and safely.

As one of state-of-the-art optimal trajectory generation methodologies, Nonlinear Trajectory Generation (NTG) combines with B-spline, nonlinear programming, differential flatness technique to generate optimal trajectories for modelled mechanical systems. While Discrete Mechanics and Optimal Control (DMOC) is a newly proposed optimal control method for mechanical systems, it is based on direct discretization of Lagrange-d'Alembert principle. In this dissertation, NTG is utilized to generate trajectories for an underwater glider with a 3D B-spline ocean current model. The optimal trajectories are corresponding well with the Lagrangian Coherent Structures (LCS). Then NTG is utilized to generate 3D opportunistic trajectories for a JPL (Jet Propulsion Laboratory) Aerobot by taking advantage of wind velocity. Since both DMOC and NTG are methods which can generate optimal trajectories for mechanical systems, their differences in theory and application are investigated. In a simple ocean current example and a more complex ocean current model, DMOC with discrete Euler-Lagrange constraints

generates local optimal solutions with different initial guesses while NTG is also generating similar solutions with more computation time and comparable energy consumption. DMOC is much easier to implement than NTG because in order to generate good solutions in NTG, its variables need to be correctly defined as B-spline variables with rightly-chosen orders.

Finally, the MARIT (Multiple Air Robotics Indoor Testbed) is established with a Vicon 8i motion capture system. Six Mcam 2 cameras connected with a datastation are able to track real-time coordinates of a draganflyer helicopter. This motion capture system establishes a good foundation for future NTG and DMOC algorithms verifications.

TABLE OF CONTENTS

DEDICATION iii

ACKNOWLEDGMENTS v

ABSTRACT vii

LIST OF TABLES xii

LIST OF FIGURES xvi

CHAPTER

I. INTRODUCTION 1

- A. Motivation 1
- B. Literature Review 5
- C. NTG and DMOC 8
- D. Dissertation Contributions 9
- E. Dissertation Outline 10

II. GLIDER TRAJECTORY GENERATION WITH NTG 13

- A. Problem Definition 13
- B. Glider Trajectory Generation 14
- C. Ocean Current Model 16
  - 1. 2D B-spline Ocean Flows Model 16
  - 2. 3D B-spline Ocean Flows Model 17
- D. Nonlinear Trajectory Generation 20
  - 1. Cost Function 20
  - 2. Constraints 21
- E. Optimal Control of a Kinematical Glider 21
  - 1. NTG Solution for 3D B-spline Ocean Flows Model 22
  - 2. Comparison in the 2D AND 3D B-spline Ocean Current Models 24
    - a. Trajectory to Minimize the Energy Only 24
    - b. Trajectory to Minimize the Energy and Time 25

F. Optimal Control of a Dynamical Glider 27
  1. Cost Function 27
  2. Constraints 28
  3. NTG Solutions for the Dynamic Glider 29
G. Animation of Glider and Ocean Current 31
H. Summary 31
III. AEROBOT TRAJECTORY GENERATION WITH NTG 34
A. NASA-JPL Aerobot 34
B. Euler-Lagrange Based Aerobot Trajectories 38
  1. Euler-Lagrange Equations 39
  2. Wind Profile 42
  3. Problem Formulation 43
  4. Simulated 3D Trajectory 44
C. State Space Model Based Trajectories 47
  1. Problem Formulation 51
  2. Simulated 3D Trajectories 52
D. Summary 55
IV. TRAJECTORY GENERATION WITH DMOC 56
A. DMOC Methodology 56
B. DMOC Tutorial 58
  1. IPOPT 59
  2. AMPL 61
  3. Implementation Details 62
  4. An Application Example 65
C. Summary 68
V. COMPARISON OF DMOC AND NTG 69
A. Discrete Mechanics and Optimal Control 69
  1. Discrete Cost Function 70
  2. Discrete Lagrange-d'Alembert Principle 70
B. Nonlinear Trajectory Generation 71
  1. Problem Formulation 71
  2. Procedure in NTG 73
C. DMOC versus NTG 73
  1. A Glider in the Simple Current Model 74
    a. Problem Formulation in DMOC 74
    b. Problem Definition in NTG 77
    c. Comparisons 80

D. A Glider in the B-spline Ocean Model 90
E. Hovercraft Example 93
F. Summary 96

VI. THE UOFL MARIT TESTBED 98

A. The MARIT Testbed 98
B. Vicon Vision System 98
C. Real-Time Application 99
D. Summary 102

VII. CONCLUSION AND FUTURE WORK 105

A. Conclusion 105
B. Future Work 106

REFERENCES 114

APPENDIX

I. NOMENCLATURE 115

II. NTG program for a glider in a B-spline ocean model 116

III. NTG program for a JPL Aerobot 131

IV. DMOC program for a glider in a B-spline ocean model 141

V. DMOC program for a JPL Aerobot 145

VI. MATLAB program for generating ODE45 trajectories 149

VII. Program to obtain real-time coordiates of a draganflyer 152

CURRICULUM VITAE 154
A. CONTACT INFORMATION 154
B. EDUCATION 154
C. RESEARCH EXPERIENCE 154
D. JOURNAL PUBLICATIONS 155
E. CONFERENCE PUBLICATIONS 155

F. HONORS AND AWARDS 156
G. SERVICE 156

LIST OF TABLES

TABLE PAGE

1. A kinematical glider in a 3D B-spline ocean current model 24

2. A kinematical glider in the 3D B-spline ocean current model versus the 2D model. 26

3. A dynamical glider in the 3D B-spline ocean current model 30

4. 3D trajectories generated by NTG for the Euler-Lagrange based Aerobot 47

5. 3D trajectories generated by NTG for the state-space based Aerobot 55

6. DMOC solutions for a dynamical glider in the 3D B-spline current model. 67

7. DMOC vs. NTG for a glider in a simple ocean current model 81

8. DMOC solutions with different intervals with the same initial guess (Guess 3) 85

9. DMOC solutions in NTG constraints versus DMOC Constraints (EuX) 86

10. DMOC solutions in NTG constraints versus DMOC constraints (EuY) 86

11. DMOC vs. NTG for a glider in a B-spline ocean model 93

12. DMOC vs. NTG for a hovercraft trajectory 96

LIST OF FIGURES

FIGURE PAGE

1. The ocean floor survey by NOAA [1]. 2
2. The Mars Rover by NASA [2]. 4
3. Surveying Mars by an Aerobot [3]. 4
4. The Slocum glider [4]. 15
5. The ocean current data and the 2D B-spline model for time t=10 and 13 (hours). 17
6. The ocean current data and the 3D B-spline model for time t=10 and 13 (hours). 18
7. The ocean current data and the 3D B-spline model for u(x,y,t) when x is fixed at -122.3061. 19
8. The ocean current data and the 3D B-spline model for v(x,y,t) when x is fixed at -122.3061. 19
9. Trajectories of the kinematical glider in the 3D ocean current model. 23
10. Speed of the kinematical glider in the 3D ocean current model. . . . 23
11. The kinematical glider minimizing-energy trajectories in the 3D and 2D ocean current models. 25
12. A kinematical glider minimizing-energy-and-time trajectories in the 3D and 2D ocean current models. 26
13. Trajectories of a dynamical glider in the 3D ocean current model. . . 29
14. Speed of the dynamical glider in the 3D ocean current model. . . . 29
15. Orientation of the dynamical glider in the 3D ocean current model. 30

16. The figure shows the correspondence with the optimal trajectories shown in Figure 11 and an LCS. Note that the red and pink in the figures near the LCS represents the location of the AUVs while the blue represents the final target location. 32

17. The figure shows the correspondence with the optimal trajectories shown in Figure 13 and an LCS. Note that the red and pink in the figures near the LCS represents the location of the AUVs while the blue represents the final target location. 32

18. An artists view of an Aerobot exploring a planet [5]. 36

19. A JPL Aerobot for exploration of Titan and Venus [5]. 36

20. The Euler-Lagrange based Aerobot controls. 38

21. The wind profile for the state-space based Aerobot. 43

22. The minimizing-energy trajectory generated by NTG for the Aerobot. 45

23. The control input τ for the minimizing-energy trajectory. 45

24. The pitch angle ϕ for the minimizing-energy trajectory. 46

25. The minimizing-time trajectory generated by NTG for the Aerobot. 46

26. The control input τ for the minimizing-time trajectory. 46

27. The pitch angle ϕ for the minimizing-time trajectory. 47

28. The state-space based Aerobot controls [6] 48

29. The wind profile for the state-space based Aerobot. 50

30. The minimizing-energy trajectory for the state-space based Aerobot. 52

31. The elevator deflection δ_e and the thrust demand δ_T for Figure 30 . 53

32. The elevator deflection δ_a and the thrust demand δ_r for Figure 30 . 53

33. The minimizing-time trajectory for the state-space based Aerobot . 54

34. The elevator deflection δ_e and the thrust demand δ_T for Figure 33 . 54

35. The elevator deflection δ_a and the thrust demand δ_r for Figure 33 . 55

36. A DMOC procedure to solve an optimization problem. 59

37. Software requirements for IPOPT, where ALS is referred to AMPL Solver Library, BLAS represents Basic Linear Algebra Subroutines, LAPACK means Linear Algebra PACKage, one Liner solver for indefinite matrices can be MA27, MA57 or other solvers, the details is described in [7]. 60

38. Dynamical glider trajectories in the 3D B-spline current model. . . . 67

39. The DMOC solution when the initial guess is a straight line. 77

40. The control input τ in DMOC when the initial guess is a straight line. 78

41. The DMOC trajectory properties when the initial guess is a straight line. 78

42. The minimizing-energy trajectory from NTG when the initial guess is a straight line. 80

43. The control input τ in NTG when the initial guess is a straight line. 81

44. The NTG optimal trajectory properties when the initial guess is a straight line. 81

45. The trajectory from DMOC versus the one from NTG when the initial guess is a straight line. 82

46. The trajectory from DMOC when initial guess changes. 82

47. The trajectory from NTG when initial guess changes. 83

48. Trajectory from DMOC versus the one from NTG when initial guess changes. 83

49. Trajectories from DMOC versus the ones from NTG when intervals are changing. 87

50. Trajectories from DMOC in details when intervals are changing. . . 87

51. Trajectories from DMOC versus NTG with no current. 88

52. Trajectories from DMOC and NTG versus Matlab ODE45 solutions. 89

53. Trajectory from DMOC when the final velocity is fixed as 0. 90

54. The 3D ocean current B-spline model u(x,y,t) at t=13 hour [8]. 91

55. The 3D ocean current B-spline model v(x,y,t) at t=13 hour [8]. 91

56. DMOC solutions in the 3D ocean current B-spline model. 92

57. NTG solutions in the 3D ocean current B-spline model. 92

58. Hovercraft [9] 94

59. Hovercraft trajectory from DMOC and NTG. 95

60. The control input for the trajectories shown in Figure 59. 96

61. A Mcam 2 camera in the Vicon Motion Capture System. 99

62. The datastation in the Vicon Motion Capture System. 100

63. A camera interface unit in the Vicon Motion Capture System. 100

64. The workstation in the Vicon Motion Capture System. 101

65. The Vicon Motion Capture System. 102

66. The Vicon iQ 2.0 in 3D live work space. 103

67. The markers are tracked in the Vicon Motion Capture System. . . . 103

68. Real-time coordinates output in the Vicon Motion Capture System. 104

CHAPTER I

INTRODUCTION

This chapter provides a general introduction and motivation to optimal trajectory generation for unmanned vehicles. Related literature review in order to highlight some of the past development is given as well. Finally, contribution and organization of the dissertation is presented.

A. Motivation

Curiosity and exploration, question and action appear in every step of human civilization history. The past and ongoing centuries witness that the world is becoming smaller and smaller due to technical advancements in transportation and communication. Exploration fields of man have expanded from tribes, counties to countries, continents and even to deep ocean, and outer space.

On the Earth where we live, more than seventy percent of surface is covered by ocean, and many parts of the ocean have not been explored and studied in details. Some questions remain to be answered. Are there any kind of sources in the ocean for future renewable energy? Can storms be predicted by studying the behavior of ocean? Manned vehicles can help people to find some answers. However, considering the cost and the potential danger in deep sea, a better option would be to utilize unmanned autonomous robotic explorers. These vehicles will be convenient even if not necessary tools for assisting scientists to investigate these kind of problems, to which part of solutions will have a huge positive impact on the world.

An Autonomous Underwater Vehicle (AUV) known as a glider plays an important role as one of robotic explorers for ocean research. The glider offers an

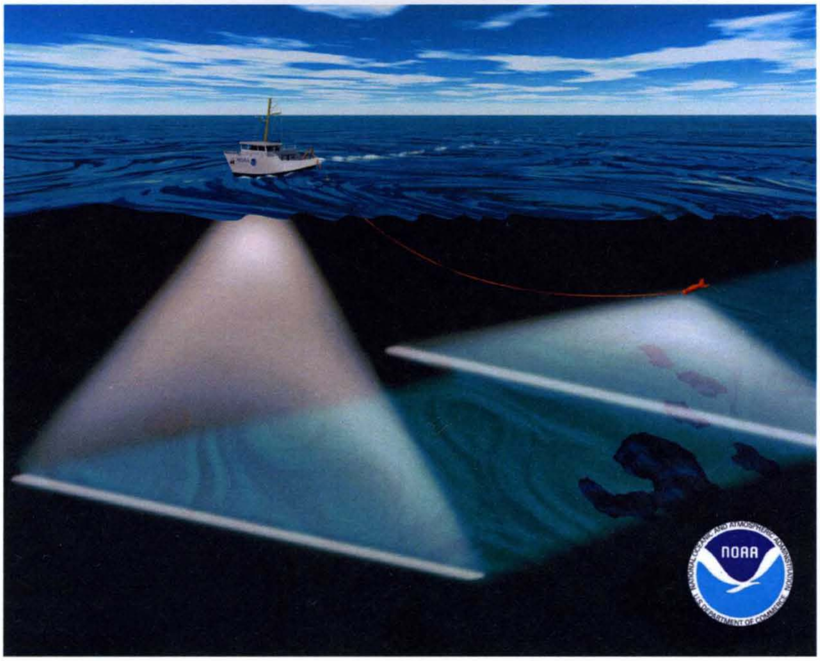


FIGURE 1 – The ocean floor survey by NOAA [1].

attractive approach for gathering data in ocean due to its relatively low cost and high sustainability. As shown in Figure 1 by NOAA (National Oceanic and Atmosphere Administration), scientists and engineers are deploying gliders for data collection. As another example of robotic explorer application, the Autonomous Ocean Sampling Network (AOSN) [10] project, [11], [12], [13] aims to advance the ability to observe and predict the ocean by bringing together sophisticated new robotic vehicles (gliders) with advanced ocean models. In the AOSN project, two types of gliders are employed, which are the *Slocum* [4], [14] and the *Spray* [15]. The gliders are designed to collect data autonomously. The efficiency and sustainability of the glider operation are important considerations for the control of the glider. Therefore, the ability to quickly determine the most efficient trajectory for the glider is important [11].

Besides deep ocean on Earth, which is just one planet in the cosmos, most spaces outside the planet or even the solar system are also mystical to humans. People started to explore space long time ago, mostly by human vision or by simple telescopes which appeared in 1608 [16]. For far beyond the limit of human vision and reachability of simple devices, what philosophers, scientists or ordinary people did most those days were imagination [17]. For instance, just before the Apollo 11 landed on the moon [18] in 1969, no one knew what characteristics the moon actually had. Many myths and stories reflected the imagination of people. Today, with years of developments in technology and science, people have the ability as well as determination to explore outer space. One might compare the current outer space exploration to the exploration of the America by the first generation of immigrants. When the pilgrims were not sure about whether they could survive to cross the Atlantic ocean, they had the courage and determination to explore the unknown. This kind of courage and determination brought about this prosperous new land. Who knows that Mars shown in Figure 2 and Figure 3 or other planets will not become new human territories for generations to come? Big achievements need to be made step by step. Using robotic explorers is the safe and cost-effective initial step for exploration. The robot rovers *Spirit* and *Opportunity* have successfully landed on the Mars. The next generation of robotic explorer such as the JPL-Aerobot can overcome some weakness of the rovers, for example, it will not get stuck by rocks or mountains, and it can explore more regions.

Both for a robotic explorer in ocean or in space exploration, trajectory generation is an essential part of its total mission planning. With optimal trajectory the exploration will become more sustainable and more efficient. The important issue is how to develop and apply an efficient trajectory generation method.

In this dissertation Nonlinear Trajectory Generation (NTG) and Discrete Mechanics and Optimal Control (DMOC) as the two state-of-the-art methodologies to generate optimal trajectory are investigated in theory and in application to an underwater glider and a JPL Aerobot. A specific kind of optimal trajectory generation problems that takes advantage of surrounding circumstances, called "Opportunistic Trajectory Generation". For the glider, ocean current flows are modelled as B-spline functions, trajectories are generated both for a kinematic glider

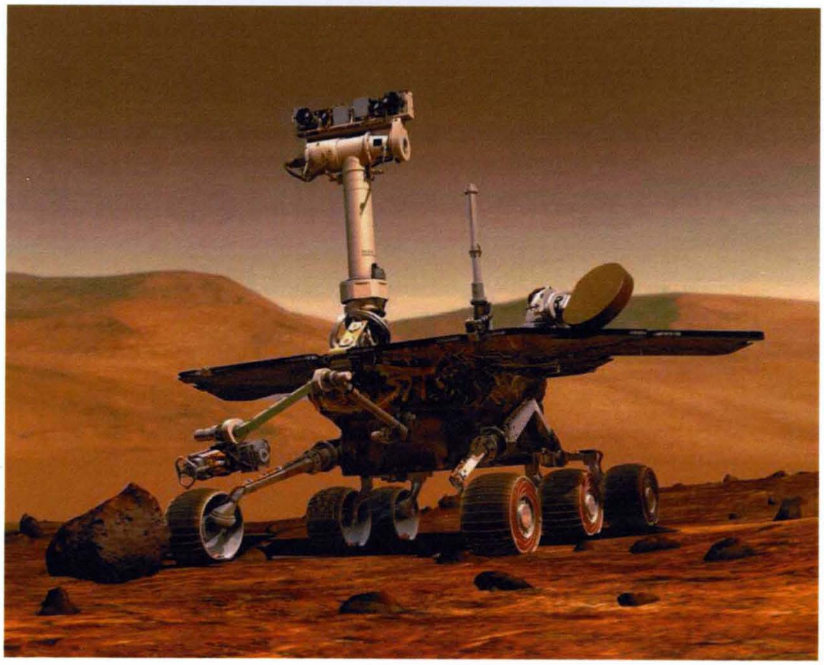


FIGURE 2 – The Mars Rover by NASA [2].

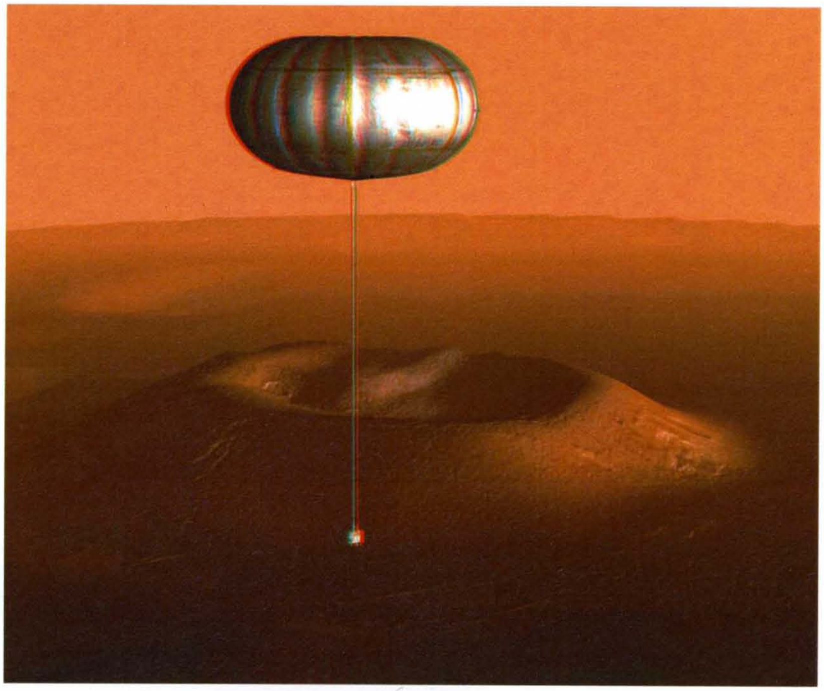


FIGURE 3 – Surveying Mars by an Aerobot [3].

and a dynamic one. The minimizing-energy trajectories are shown to correspond well with the Lagrange Coherent Structures and their energy usages are efficient. For a JPL Aerobot, NTG trajectories are generated both from the perspective of Euler-Lagrange and from the state space model with decoupled longitudinal and latitudinal dynamics. Then DMOC is introduced with a detailed theory explanation and an application procedure.

For analyzing and comparing NTG and DMOC, an underwater glider is utilized in both a simple ocean model and a complex B-spline ocean current model. The cost functions and constraints in NTG are the same as the ones in DMOC while the NTG ones have continuous Euler-Lagrange equations, DMOC ones have their discrete forms. DMOC is shown to have less computation time than NTG with the comparable energy cost. It is much easier for DMOC to model the problem and generate solutions, while on NTG, its variables should be correctly defined as B-spline functions with right orders. Finally, for the future research, a MABIT (Multiple Air Robotics Indoor Testbed) testbed with Vicon 8i vision system is being successfully established with the proposed program to track real-time coordinates of draganflyers [19].

B. Literature Review

Optimal control as a research topic came into being in June 1696 when Professor Johann Bernoulli published his solution to the Brachistochrone ("shortest time" in Greek) problem [20]. This problem as a challenge in 1696 caught the attention of giants like Newton, Leibniz, Tschirnhaus, l'Hopital and Jakob Bernoulli who published their solutions in May 1697. However this kind of problem is not systematically solved. Only after years of development, in 1744, as a student of Bernoulli, Euler gave a general procedure for writting down what later became known as Euler's equations in his book "the Method of finding Plane Curves that Show Some Property of Maximum and Minimum". About ten years later, Lagrange eliminated the tedium and need for geometrical insight in Euler's method and attained the same solution by analysis alone. He derived the Euler-Lagrange equation for the necessary first variation condition of optimum. The standard optimization problem is shown in (1), its Euler-Lagrange equation is presented in (2).

$$minimize\ J = \int_{t_0}^{t_f} L(q(t), \dot{q}(t), t)dt, subject\ to\ q(t_0) = A, q(t_f) = B \tag{1}$$

$$\frac{d}{dt} \cdot \frac{\partial L}{\partial \dot{q}} = \frac{\partial L}{\partial q} \tag{2}$$

The second variation as an additional necessary condition for a minimum was done by Legendre (1752-1833). Legendre's condition for the scalar case is in (3), while the Hessian matrice has to be nonnegative definite for the vector case.

$$\frac{\partial^2 L}{\partial \dot{q}^2} > 0 \tag{3}$$

Hamilton wrote Hamiltonian to simplify the previous Euler-Lagrange equations. Around 1836, Hamilton and Jacobi showed that the partial derivatives of the performance index with respect to each parameter of a family of extremely obeyed the Hamilton-Jacobi equation. This is the basis of Dynamic Programming proposed by Bellman over 100 years later. Weierstrass derived the side condition which can transform the minimization problem into Weierstrass's form, it is the predecessor of the maximum principle [21]. In the middle of 20th century, optimal control was basically developed due to the maximum principle by L. S. Pontryagin [22] and the dynamic programming method by R. Bellman [23]. Compared to the maximal principle by Pontryagin, the method of dynamic programming was developed for the needs of optimal control processes which are of a much more general character than those which are described by systems of differential equations. Therefore, the method of dynamic programming carries a more universal character than the maximum principle, but it does not have the rigorous logical basis but a heuristic method. Some assumptions are needed to derive Bellman's equations which even in the simplest examples do not hold. In the 1960s

Kalman [21] et.al. showed that the MIMO (Multi Input and Multi Output) LQ (Linear Quadratic) optimal control problem can be solved numerically and efficiently with a backward sweep of a matrix Ricatti equation. He introduced the concept of state and control variables and proposed a compact vector-matrix notation which became standard in optimal control. To solve optimal control problems numerically, the paper [24] proposed that control states can be approximated by values at a finite number of time points, the control history can be parametrized by piecewise polynomials, and further this problem can be solved by a standard Nonlinear Programming solver. The idea is quite similar to what NTG implements in problem formulation. Nonlinear Trajectory Generation (NTG) is based on a combination of spline function (piecewise polynomials), nonlinear programming and differential flatness. Discrete Mechanics and Optimal Control is based on direct discretization of states and direct discretization of Lagrange-d'Alembert principle which results into discrete Euler-Lagrange equation.

Optimal trajectory generation problem is one kind of optimal control problems. Its applications vary from the control of various devices such as control of linear system [25], and engine valves [26], to motion planning of robots [27] [28], manipulator robots [29] [30], humanoid robot [31], and trajectory tracking for boom cranes [32]. Optimal trajectory generation for hypersonic vehicles as a research topic was raised in [33], Philip D. Hattis and Richard K. Smolskis proposed a calculus of variations direct method of steepest descent to determine the trajectory for hypersonic vehicles. The trajectory optimization algorithm is based on a gradient/steepest descent technique for solving two point boundary value problems, however this method has little hope of being realizable as a real-time algorithm. To make the problem solvable in real-time, Nonlinear Trajectory Generation can also exploit the possible differential flatness of the system to speed up the computation time and parametering trajectory with B-spline [34] functions, while DMOC efficiency is shown later impressive by directly discretizing Lagrange-d'Alembert principle without first deriving equations of motion to generate optimal solutions.

Particularly, opportunistic [35] [36] [37] trajectory generation on which this dissertation focused is taking advantages of circumstances such as current or wind velocities, to generate optimal trajectories for robotic explorers.

C. NTG and DMOC

In this section, NTG and DMOC as two different state-of-the-art methodologies to solve optimal control problems for mechanic systems are introduced.

Milam [38] et al. developed NTG which is designed to generate real-time trajectories. "Real-Time" means the method should generate a solution fast enough for a real-time application. The optimal trajectory generation is generally concluded as a nonlinear programming problem. For nonliner programming, if the problem scale is large and complex, it is not straightforward to get the solution in real-time. Therefore, the real-time optimal trajectory generation is a challenging task. As it is indicated in [38], some standard numerical solution of optimal control problem cannot be implemented in real-time. NTG use the nonlinear geometric control [39] techniques to solve the optimal control problem much faster than the standard method. This technique can first exploit differential flatness [40] of the system to reduce the complexity of the problem then use the collocation method to solve the optimization problem. In NTG the variables are represented in the format of B-spline [34] functions.

Discrete Mechanics and Optimal Control (DMOC) is developed by Jerrold E. Marsden et. al., which is presented in [9]. Basically, the system states, control forces, equality constraints are discretized based on the direct discretization of the Lagrange-d'Alembert principle. The expected key advantages over traditional methods are less energy consumption for system control purpose and more robust to modelling errors [9]. After discretization, the resulted finite dimensional nonlinear optimal control problem is also solved by the sequential quadratic programming (SQP) [41].

NTG and DMOC are two different approaches which can be utilized to solve optimal trajectory generation problems of robotic explorers. The complete evaluation of differences between two methodologies is one of the main parts of this dissertation.

D. Dissertation Contributions

This dissertation investigates two state-of-the-art optimal trajectory generation methods NTG and DMOC, their theoretic and practical differences are presented with applications to an underwater glider and a JPL aerobot. The contributions of this dissertation work are listed in the following:

First, optimal trajectory generation for an underwater glider is presented, in this part, ocean flows are modelled by 3D B-spline functions since NTG need derivatives of variables in its problem formulation. Both for a kinematic and dynamic glider, trajectories with NTG are successfully obtained, which are corresponding well with Lagrangian Coherent Structures (LCS). They are shown by animations that LCS and NTG can generate optimal trajectories for a glider to save the energy with known ocean current velocities.

Secondly, for a robotic explorer in the space, a JPL Aerobot is modelled with consideration of its aerodynamics, and constraints as Euler-Lagrange equations. NTG successfully generates 3D optimal trajectories for minimizing energy and minimizing time in a defined wind field. Furthermore, a decoupled longitudinal and lateral dynamics of an Aerobot state-space model is also utilized to generate optimal trajectories. The solutions are energy efficient from NTG.

Thirdly, Discrete Mechanics and Optimal Control as a methodology for solving optimal control problems for mechanic systems are presented with its adaption to solve the trajectory generation problems. The problem is modelled by AMPL, the solver is chosen as IPOPT, this dissertation presents a detailed procedure to use the available tools with DMOC to solve optimal trajectory generation problems for mechanical systems.

Fourthly, DMOC and NTG are compared with a glider in a simple ocean current model and a B-spline ocean current model. In a simple ocean current example, DMOC with discrete Euler-Lagrange constraints generates local optimal solutions with different initial guesses while NTG is also generating similar solutions with more computation time and comparable energy consumption. Further in a more complex ocean current model, optimal solutions from DMOC also cost similar energy and computation time than the ones from NTG. The cost functions are the integral of gyroscopic forces over time, nonlinear constraints are direct discrete Euler-Lagrange equations for DMOC, continuous ones for NTG.

Finally, an Unmanned Air Vehicle 3D testbed is preliminarily established in our lab, a Vicon 8i motion capture system with 6 Vicon MCam 2 cameras is utilized in the system to track real-time coordinates of a draganflyer helicopter. A c++ program is written to connect the real-time engine of the Vicon system with the user program, the draganflyer with markers can be modelled as a rigid body, the proposed program has the ability to retrieve the 6 DOF (Degree of Freedom) information. It makes a good foundation to further utilize this testbed to test control or planning algorithms such as NTG and DMOC for the UAVs.

E. Dissertation Outline

This dissertation consists of seven chapters, with seven appendices. The dissertation is organized as follows.

1. **Chapter I** is the introductory part of the whole dissertation. The background and motivation of optimal trajectory generation for ocean and space exploration are introduced and discussed. A literature review is provided to show the work in the context of optimal control research history. Then, Nonlinear Trajectory Generation (NTG) and Discrete Mechanics and Optimal Control (DMOC) as two optimal trajectory genera-

tion methodologies are introduced.

2. **Chapter II** is showing the work to generate optimal trajectory for an underwater glider with NTG. In this Chapter, ocean current data is modelled as B-spline functions, the minimizing-energy trajectories are shown to corresponds with Lagrange Cohere Structures (LCS).
3. **Chapter III** generates opportunistic trajectories for a JPL Aerobot with NTG. In this Chapter, wind profile data is modelled as layers, the aerobot with simplified model from Euler-Lagrange perspective and with state-space based model are both investigated. The control inputs are easily appliable. The minimizing-energy trajectories are shown to take advantage of wind velocities more than the minimizing-time trajectories. It is shown that energy-efficient trajectories can be generated based on NTG methodology.
4. **Chapter IV** presents a new DMOC approach of real time trajectory generation method. The detailed procedure is provided to use DMOC approach with AMPL and IPOPT to solve optimal control problems for mechanical systems. It is shown that user-defined functions can be involved to solve more complex problems in DMOC problem formulation.
5. **Chapter V** compares the DMOC trajectory generation method with the NTG method with application to an underwater glider both in a simple ocean current model and a B-spline ocean current model. The results show that DMOC is easy to implement, cost less computation time and comparable energy cost than NTG.
6. **Chapter VI** describes the procedure to upgrade the U of L mobile robot testbed to a 3D UAV (Unmanned Air Vehicle) testbed with a Vicon motion capture system and draganflyer helicopters. It is shown that the newly established UAV testbed can obtain 6 DOF information of a defined rigid body in real-time. It makes a good foundation to utilize this testbed for future NTG and DMOC algorithms verifications experimen-

tally.

7. **Chapter VII** concludes the dissertation with clarifying the main contents of this dissertation and directions for further research.

CHAPTER II

GLIDER TRAJECTORY GENERATION WITH NTG

A. Problem Definition

Optimal trajectory generation for a glider can be considered as one kind of optimal control problems. Consider a general dynamical system [11] [8] which includes a glider under investigation:

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{u}(t)) \tag{4}$$

where $\mathbf{x}(t)$ is the state of the system and $\mathbf{u}(t)$ is a control input. For optimal control, given a cost function of the form:

$$J = \Phi_0(\mathbf{x}(t_0), \mathbf{u}(t_0), t_0) + \int_{t_0}^{t_f} L(\mathbf{x}(t), \mathbf{u}(t), t)dt + \Phi_f(\mathbf{x}(t_f), \mathbf{u}(t_f), t_f) \tag{5}$$

It is suitable to choose $\mathbf{u}(t)$ for $t \in [t_0, t_f]$ which minimizes J subject to constraints of the form

$$\begin{array}{llcll} \text{Initial} & lb_0 & \leq & \Psi_0(\mathbf{x}(t_0), \mathbf{u}(t_0), t_0) & \leq & ub_0 \\ \text{Trajectory} & lb_t & \leq & \Psi_t(\mathbf{x}(t), \mathbf{u}(t), t) & \leq & ub_t \\ \text{Final} & lb_f & \leq & \Psi_f(\mathbf{x}(t_f), \mathbf{u}(t_f), t_f) & \leq & ub_f \end{array} \tag{6}$$

Notice that the cost function J is composed of an initial condition cost, $\Phi_0(\cdot)$, an integral cost over the trajectory, $L(\cdot)$, and a final condition cost, $\Phi_f(\cdot)$. The constraints are similarly partitioned. lb and ub represent lower and upper bounds, respectively. Cost (5) and (6) are standard in optimal control, and further explained in [42] and [43]. An optimal solution for a specified problem is obtained generally by nonlinear programming. After the optimal control problem with costs and constraints are modelled, it can be expressed mathematically as nonlinear programming problem in which a solver is required. In NTG the nonlinear programming solver is NPSOL [44] which is developed by Philip Gill et. al. NPSOL employs a dense Sequential Quadratic Programming (SQP) [41] algorithm and the user must supply an initial guess of the solution to the problem, and define subroutines that evaluate cost and constraint functions. If the problem is large and sparse, MINOS [45] package should be used, since NPSOL treats all matrices as dense. If there are not nonlinear constraints, gradients of the bound and linear constraints are never recomputed, and NPSOL will function as a specialized algorithm for a linearly constrained optimization problem. It can be arranged that the problem functions are evaluated only at points that are feasible with respect to bounds and linear constraints. NPSOL uses subroutines from the LSSOL [46] constrained linear least squares package, which is distributed together with NPSOL.

B. Glider Trajectory Generation

Autonomous Underwater Vehicles (AUVs) including gliders are becoming more and more popular [47], [48], [49]. For example, oil companies can use gliders to make a detailed underwater map or search resources before they decide a next step to exploit. Besides industry application, some research related projects also need to use this kind of robotic explorer. The Autonomous Ocean Sampling Network II project (AOSN-II) [10], [50], [11], [12], [13] aims to advance the ability to observe and predict the ocean by bringing together sophisticated new robotic vehicles (gliders) with advanced ocean models. In this project, two types of gliders are employed, which are the SLOCUM [4], [14] and the SPRAY [15].

Gliders offer an attractive means for gathering data in the ocean because they are relatively low cost and highly sustainable. For adaptive ocean sampling, the gliders are often redirected throughout the ocean to areas of high uncertainty or transient features of interest. Therefore the ability to quickly determine the most efficient trajectory for a glider to take is desirable. It is also necessary to minimize the glider energy usage in order to keep it autonomously operational for the great-

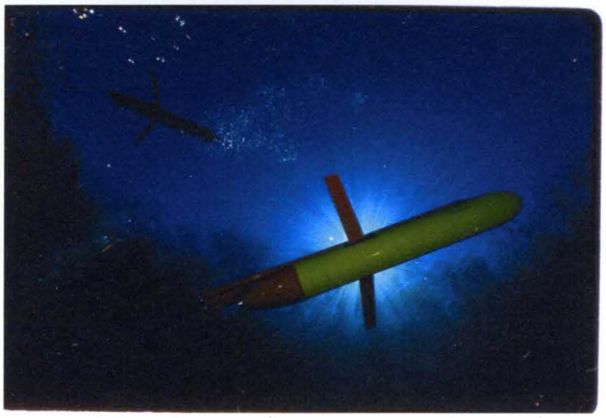


FIGURE 4 – The Slocum glider [4].

est amount of time. The tradeoff for a glider's remarkable efficiency with the modest energy cost is a relatively low average speed for the vehicle. Typically, gliders move around 40 (cm/s) relative to the ambient water. However, the ambient water can often move at speeds the same order of magnitude as the speed of the glider. For instance, in Monterey Bay, CA, which was the location for the AOSN-II experiment, the surface currents average velocity is around 20 (cm/s), and it is typically stronger outside the bay. Therefore it is advantageous, if not necessary, to make use of ocean currents to help propel the gliders around the ocean for sustainable missions.

This chapter is to extend the previously proposed method [11] for quickly determining near optimal glider trajectories between two fixed points in the ocean based on approximate ocean current data. It will show that optimal trajectories computed using NTG corresponds to LCS obtained using the Direct Lyapunov Exponent method [51]. There are two parts are tackled in this chapter. One is to improve the previous analytical ocean flows model [11], which is required in the NTG formulation, to a 3D model using B-spline functions. The other is to establish a new dynamical model of the glider. Then, these models are used in the NTG to find near optimal trajectories for the glider.

The ocean flows velocity data used for these computations was obtained from High Frequency Radar stations measuring surface currents around the Monterey Bay, CA [52] and processed by Open-Boundary Modal Analysis [53] to smooth the data and fill in the missing data points. In the NTG formulation [38], [11], the costs and the constraints in terms of outputs and their derivatives need to be specified. As will be seen in the following sections that ocean flows velocity field will appear in the costs and constraints of the optimal control problem. Therefore, the NTG method needs the derivatives of the velocity field with respect to the outputs. Numerically computing these derivatives directly from the velocity data sets can easily create convergence problems. Thus, it is better to use approximation techniques to find a smooth analytical model for the data. For this, the B-spline functions are employed, allowing straightforward computation of derivatives.

C. Ocean Current Model

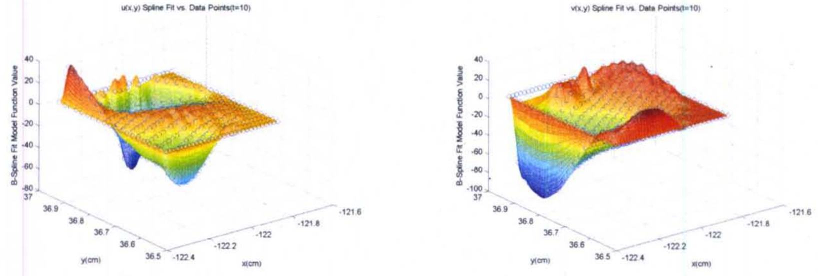
1. 2D B-spline Ocean Flows Model

B-splines are commonly used in data approximation and calculation [34]. In the previous work [11], ocean current flows are modeled using 2D B-spline function as given below:

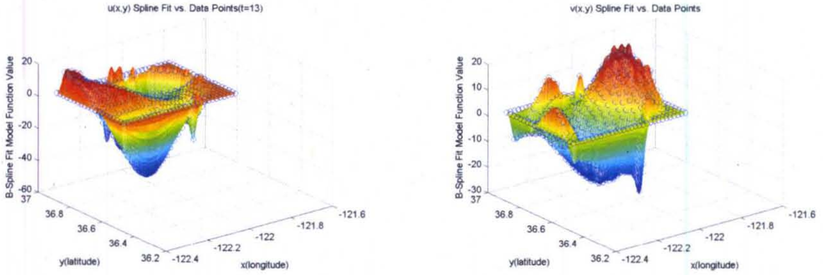
$$\begin{array}{rcl} u(x,y) & = & \sum_{i=1}^{m}\sum_{j=1}^{n} B_{i,k_{ux}}(x)B_{j,k_{uy}}(y)a_{ij} \\ v(x,y) & = & \sum_{i=1}^{p}\sum_{j=1}^{r} B_{i,k_{vx}}(x)B_{j,k_{vy}}(y)b_{ij} \end{array} \tag{7}$$

where a_{ij} and b_{ij} represent coefficients of the B-spline function for $u(x,y)$ and $v(x,y)$ which are components of the ocean currents in the x- and y-direction, respectively. Coordinates are chosen such that the x-axis is in the direction of increasing longitude and the y-axis in the direction of increasing latitude. $B_{i,k}$ and $B_{j,k}$ represent B-spline basis functions for the x- and y- direction, respectively. The orders of the polynomials were $k_{ux} = k_{uy} = k_{vx} = k_{vy} = 4$ and the numbers of the coefficients were $m = p = 32$ and $n = r = 22$.

The parametrizations given by (II.C.1), developed in the previous work [11],



a) u(x,y) model,t=10 b) v(x,y) model,t=10



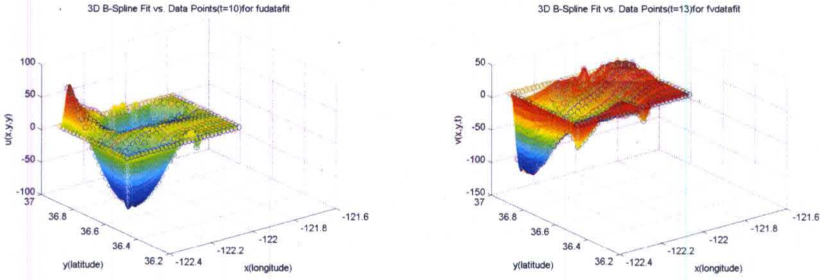
c) u(x,y) model,t=13 d) v(x,y) model,t=13

FIGURE 5 – The ocean current data and the 2D B-spline model for time t=10 and 13 (hours).

do not incorporate the time dependence of the currents. The time dependence of the velocity data was built into the NTG by assuming that the velocity fields were constant over hourly intervals. For every hour a different ocean model was calculated. Then, these models were used in a receding-horizon approach where at every hour a new trajectory was calculated from the current location of the glider to the final destination. Figure 5 shows $u(x, y)$ and $v(x, y)$ from ocean current data and a 2D B-spline model at times $t = 10$ and $t = 13$ hours.

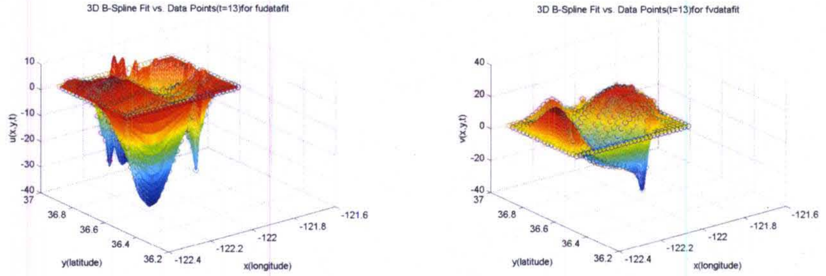
2. 3D B-spline Ocean Flows Model

Further in this section, the 2D ocean current flows model is extended to a 3D B-spline model incorporating the time dependence of the currents explicitly as shown in (8).



a) u(x,y,t) model,t=10

b) v(x,y,t) model,t=10



c) u(x,y,t) model,t=13

d) v(x,y,t) model,t=13

FIGURE 6 – The ocean current data and the 3D B-spline model for time t=10 and 13 (hours).

$$\begin{aligned} u(x,y,t) &= \sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{k=1}^{o} B_{i,k_{ux}}(x)B_{j,k_{uy}}(y)B_{k,k_{ut}}(t)a_{ijk} \\ v(x,y,t) &= \sum_{i=1}^{p}\sum_{j=1}^{r}\sum_{k=1}^{s} B_{i,k_{vx}}(x)B_{j,k_{vy}}(y)B_{k,k_{vt}}(t)b_{ijk} \end{aligned} \tag{8}$$

where a_{ijk} andb_{ijk} represent coefficients of B-spline for u and v,respectively. $B_{i,k}$, $B_{j,k}$ and $B_{k,k}$ represent B-spline basis functions for the x- , y- and t- direction, respectively. The orders of the polynomials used were $k_{ux} = k_{uy} = k_{vx} = k_{vy} = k_{ut} = k_{vt} = 4$ and the numbers of the coefficients were $m = p = 32$, $n = r = 22$ and $o = s = 25$.

The 3D B-spline ocean model has three input variables-longitude, latitude and time. In order to visualize the model, the time is fixed as it is in 2D function. The results in Figure 6 are similar with the 2D case results shown in Figure 5 as expected.

Figure 7 and Figure 8 show the 3D B-spline ocean flows model changing with time where x-direction is fixed at -122.3061 (deg) for ease of visualization purposes only.

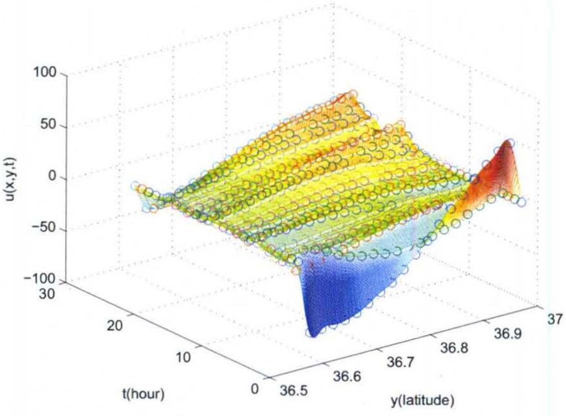


FIGURE 7 – The ocean current data and the 3D B-spline model for u(x,y,t) when x is fixed at -122.3061.

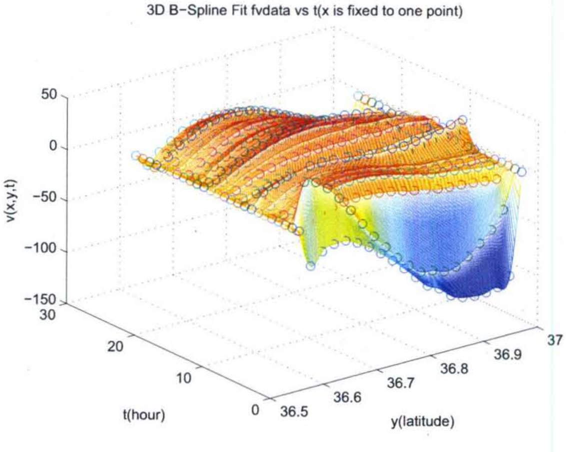


FIGURE 8 – The ocean current data and the 3D B-spline model for v(x,y,t) when x is fixed at -122.3061.

D. Nonlinear Trajectory Generation

Nonlinear Trajectory Generation (NTG) developed by Milam et al. [38], [44] is designed to solve constrained nonlinear optimal control problems in real time. The main advantage of NTG compared to other dynamic optimization methods is that it can quickly provide sub-optimal solutions, which makes it very attractive for real-time application. In addition, linear as well as nonlinear constraints and cost functions can be defined in the problem formulation of NTG.

NTG is based on a combination of nonlinear control theory, spline theory and sequential quadratic programming. With the optimal control problem formulation, characterization of trajectory space, and collocation points definition, NTG transforms the optimal control problem into a Nonlinear Programming (NLP) problem solved by NPSOL [44], a popular NLP solver, which uses Sequential Quadratic Programming (SQP). The baseline NTG algorithm has been described extensively in the literature [43], [38], [11], therefore in this section it is outlined briefly.

1. Cost Function

The cost function for this problem is a weighted sum of a time cost and an energy cost as follows:

$$J \ = W_t T + \int_0^1 W_u \left(\left(\frac{\dot{x}}{T} - u \right)^2 + \left(\frac{\dot{y}}{T} - v \right)^2 \right) T d\tau$$

$$\dot{\mathbf{x}} = \frac{dx}{d\tau} \tag{9}$$

$$\dot{\mathbf{y}} = \frac{dy}{d\tau} \tag{10}$$

Where W_t and W_u represent the weighting on the total mission time and energy expenditure, respectively. Note that the T terms in the integral, representing the unknown final mission time, and the integral bounds ranging from 0 to 1 are both due to introducing time as a state variable in the NTG formation which is not straight forward. This is explained in detail in [11].

2. Constraints

Constraint functions are given as [11]:

- (Linear) Initial Constraints:

$$-122.1780 - \epsilon(\text{deg}) \leq x(0) \leq -122.1780 + \epsilon(\text{deg})$$

$$36.8557 - \epsilon(\text{deg}) \leq y(0) \leq 36.8557 + \epsilon(\text{deg})$$

$$0 \leq T \leq 48 \; hours$$

- (Linear) Final Constraints:

$$-122.2420 - \epsilon(\text{deg}) \leq x(T) \leq -122.2420 + \epsilon(\text{deg})$$

$$36.6535 - \epsilon(\text{deg}) \leq y(T) \leq 36.6535 + \epsilon(\text{deg})$$

- (Nonlinear) Trajectory Constraints:

$$1 \leq W_v \frac{1}{T^2}\left(\left(\frac{dx}{d\tau}\right)^2 + \left(\frac{dy}{d\tau}\right)^2\right) \leq 1600$$

E. Optimal Control of a Kinematical Glider

The optimal control problem considered here is to find optimal glider trajectories, -in the case of time, or energy, or time and energy -, between two fixed points in the ocean utilizing the NTG method. The same start and destination points as in [11] are used for comparison:

$$\begin{aligned} (\mathbf{x}(t_0), \mathbf{y}(t_0)) &= (-122.178(\text{deg}), 36.8557(\text{deg})) \\ (\mathbf{x}(t_f), \mathbf{y}(t_f)) &= (-122.242(\text{deg}), 36.6535(\text{deg})) \end{aligned} \tag{11}$$

In order to compare the 2D B-spline ocean flows model with the 3D B-spline model, first the 2D kinematical glider model as in [11] considered:

$$\begin{aligned} \dot{x} &= V\cos\theta + u \\ \dot{y} &= V\sin\theta + v \end{aligned} \tag{12}$$

where V is the speed of the glider, θ is the orientation of the glider, $u(x, y, t)$ and $v(x, y, t)$ are the components of the ocean currents in the x and y direction, respectively.and V is a control input. The pair $(u(x, y, t), v(x, y, t))$ is referred to as the (time-dependent) velocity field.

1. NTG Solution for 3D B-spline Ocean Flows Model

After the 3D B-spline ocean current flows model is applied in the NTG, several optimal trajectories of the kinematical glider are obtained. The output of NTG is defined as the position sequence of the glider trajectory. The properties of the trajectories are listed in TABLE 1. In this table, *min E, min TE* and *min T* represent minimizing the energy, time and energy and time, respectively. T_f is the final mission time for the glider to travel from the start point to the final point. *Time* represents the actual running time of the NTG algorithm to find the (near) optimal solution. *Energy Cost* is the energy of the glider to travel from start to final point and it is calculated from

$$e = \int_0^{T_f} \left(\left(\frac{dx}{dt} - u \right)^2 + \left(\frac{dy}{dt} - v \right)^2 \right) dt$$

The results are reasonable considering the purpose of the trajectories. The energy cost is the maximal when the NTG only minimizes the time, and it is the least when the NTG only minimizes the energy as expected.

The following Figure 9 shows the trajectories and the Figure 10 shows that the constraints on the glider velocities are satisfied. In these figures, red, blue and green lines correspond to the *min E, min TE*,and *min T* trajectories, respectively.

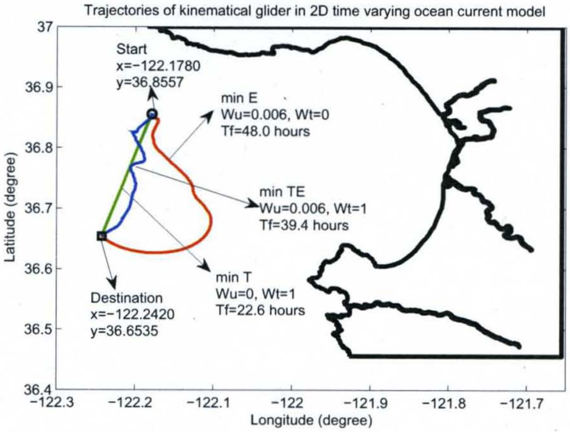


FIGURE 9 – Trajectories of the kinematical glider in the 3D ocean current model.

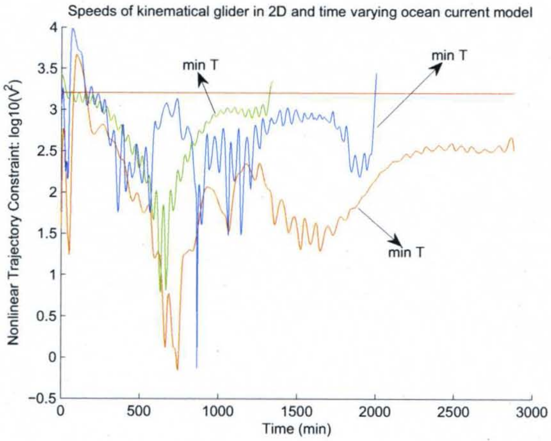


FIGURE 10 – Speed of the kinematical glider in the 3D ocean current model.

TABLE 1
A KINEMATICAL GLIDER IN A 3D B-SPLINE OCEAN CURRENT MODEL

$In\ 3D\ model$	T_f(hrs)	$Time(s)$	$Energy\ Cost(cm^2/s)$
$min\ E$	48.00	2.72	2.7538e4
$min\ TE$	39.38	1.43	9.0038e4
$min\ T$	22.60	0.5	1.3209e5

2. Comparison in the 2D AND 3D B-spline Ocean Current Models

In the following subsections, the trajectories of the kinematical glider found using 2D and 3D B-spline ocean models are compared. These two types of ocean current flows models are applied into NTG with the same kinematical glider model, refer to (12), cost and constraint functions.

a. Trajectory to Minimize the Energy Only The 2D B-spline ocean current model given by (II.C.1), does not incorporate the time dependence of the currents and it assumes that the ocean velocity fields are constant over hourly interval. Therefore, for every hour a different ocean model was found. Then, optimal trajectories were found for each hour by updating the ocean models between the current location of the glider after one hour (at time zero, this is the start point) and the final desired destination. This is to be the receding-horizon approach [11]. This causes unnecessary running of the NTG algorithms many times. On the other hand, the 3D B-spline ocean current model integrates the time into the ocean model continuously by extending the B-spline parametrizations in time (as well as space). Hence, for the 3D ocean current model, the ocean current is dynamically changing with the time, and the total trajectory can be easily acquired by running the NTG algorithm once.

Figure 11 shows optimal trajectories minimizing the energy by using two different ocean flows models. The dotted line shows concatenated trajectories found using 2D B-spline ocean model by running NTG algorithm several times, once for every hour. The solid blue line shows the glider trajectory found using

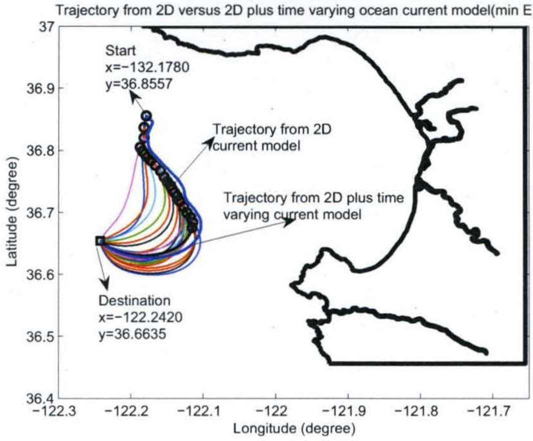


FIGURE 11 – The kinematical glider minimizing-energy trajectories in the 3D and 2D ocean current models.

the 3D B-spline ocean model.

The trajectory properties of kinematical glider in 3D ocean current model and 2D model are listed in TABLE 2. In this table, parameters are same as in the TABLE 1. Two set of values are given for *min E* and *min TE*. As it is indicated in TABLE 2, trajectories found by 3D ocean models have less energy cost than the ones found for the 2D ocean models. Specifically, for *min E* the energy cost for 3D is $2.7538e4$ while it is $4.039e5$ for 2D case. Similar results are shown for *min TE* trajectories. Another advantage of utilizing the 3D ocean models is to reduce the computation of the optimal (or near optimal) trajectories. In detail, the total execution time of NTG algorithm are 2.72 seconds for the 3D ocean models while it is 64.42 seconds for the 2D case as shown in TABLE 2 for min E. These results changes from min T as 1.43 seconds for 2D case while 42.77 seconds for the 3D case.

b. Trajectory to Minimize the Energy and Time The trajectories to minimize the energy-and-time from two models are illustrated in Figure 12. Figure 11 indicates that the minimizing-energy trajectories from 3D and 2D ocean current models are almost the same. They are very similar to each other, so it shows that the assumption in [11] that the velocity fields are constant over hourly intervals is

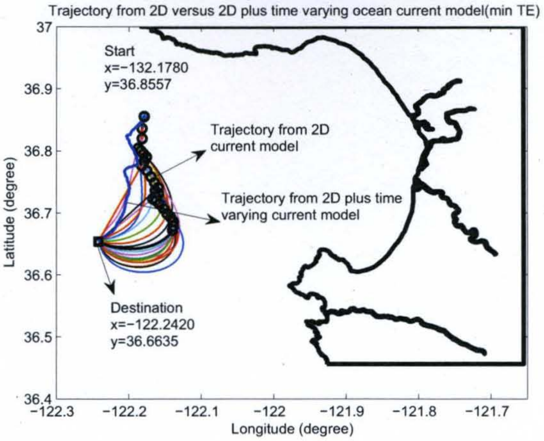


FIGURE 12 – A kinematical glider minimizing-energy-and-time trajectories in the 3D and 2D ocean current models.

TABLE 2
A KINEMATICAL GLIDER IN THE 3D B-SPLINE OCEAN CURRENT MODEL VERSUS THE 2D MODEL.

$3D/2D$	$T_f(hrs)$	$Time(s)$	$Energy\, Cost(cm^2/s)$
$min\, E$	$48.00/45.84$	$2.72/64.42$	$2.7538e4/4.039e5$
$min\, TE$	$39.38/39.31$	$1.43/42.77$	$9.0038e4/4.178e5$

tolerable in this minimizing energy case.

However, the minimizing-time-and-energy trajectories obtained from 3D and 2D ocean models are not the same as clearly shown in Figure 12. The reason is that for minimizing time-and-energy when the start point and velocity field are different, the glider might decide to choose a different way based on the current flows and the position. It will not necessarily move with the direction of the ocean flow as in the *min E* case. Even though the trajectories are different in the case of minimizing energy and time, the shapes and curves of these two trajectories are still similar with each other. Another point to remember is that, for the 2D ocean model, the trajectory is recalculated for every hour. At the end of each hour, the glider final point is taken as a start point for the new trajectory calculation.

Note that for minimizing time only trajectories, they are both straight lines for 2D and 3D ocean models since the ocean current speed is not large enough against the forward direction of the glider.

F. Optimal Control of a Dynamical Glider

The trajectories of the glider with the kinematical model are already obtained. In this section, dynamics of the glider is taken into account in purpose of producing more realistic glider trajectories. The glider is assumed to be actuated by a gyroscopic force F_{gyr} which implies that the relative forward speed of the glider is constant. However, the orientation of the glider cannot change instantly and the control force is the change in the orientation of the glider. The dynamic model of the glider is presented in the following:

$$\begin{aligned} \ddot{\mathbf{x}} &= -V\tfrac{d\theta}{dt}\sin\theta + \dot{u} \\ \ddot{\mathbf{y}} &= V\tfrac{d\theta}{dt}\cos\theta + \dot{v} \end{aligned} \tag{13}$$

According to (12), then the dynamical glider model can be expressed as:

$$\begin{aligned} \ddot{\mathbf{x}} &= -\tfrac{d\theta}{dt}\left(\dot{y} - v\right) + \dot{u} \\ \ddot{\mathbf{y}} &= \tfrac{d\theta}{dt}\left(\dot{x} - u\right) + \dot{v} \end{aligned} \tag{14}$$

The gyroscopic force is given by:

$$F_{gyr} = \begin{pmatrix} -\dfrac{d\theta}{dt}\left(\dot{y} - v\right) \\ \dfrac{d\theta}{dt}\left(\dot{x} - u\right) \end{pmatrix} \tag{15}$$

The gyroscopic force acts proportional to the relative velocity between fluid and the glider.

1. Cost Function

For the dynamic glider model, the control force is the F_{gyr}. Therefore, the cost function is changed from (9) to the following:

$$J \ = W_tT + W_u \int_0^T \ \| F_gyr \|^2 \, dt$$

The cost function can be further expressed by utilizing (13) and (15), as:

$$J \ = W_tT + W_u \int_0^T \ \left((\ddot{x} - \dot{u})^2 + (\ddot{y} - \dot{v})^2 \right) dt$$

$$\theta = \frac{t}{T} \tag{16}$$

Then, the cost function is obtained for the dynamic glider after introducing the time as a state variable in the NTG formulation [11] using16 as:

$$J \ = W_tT + W_u \int_0^1 \left(\left(\frac{\ddot{x}}{T^2} - \dot{u} \right)^2 + \left(\frac{\ddot{y}}{T^2} - \dot{v} \right)^2 \right) T d\tau$$

2. Constraints

The constraints for the dynamic glider model are almost the same as the ones in the kinematical glider model in the previous section. One more constraint related with the control force F_{gyr} is added since it cannot be infinitely large. Therefore, the constraints for the glider orientation change are introduced as shown in (17):

$$-18 \, (\deg / s) \leq \tfrac{d\theta}{dt} \leq 18 \, (\deg / s) \tag{17}$$

The constraint function (17) can be further expressed utilizing (12) and (14), as:

$$-18 \, (\deg / s) \leq \tfrac{\ddot{y} - \dot{v}}{\dot{x} - u} \leq 18 \, (\deg / s) \tag{18}$$

where

$$\dot{x} = dx/dt, \ddot{y} = d^2y/dt^2$$

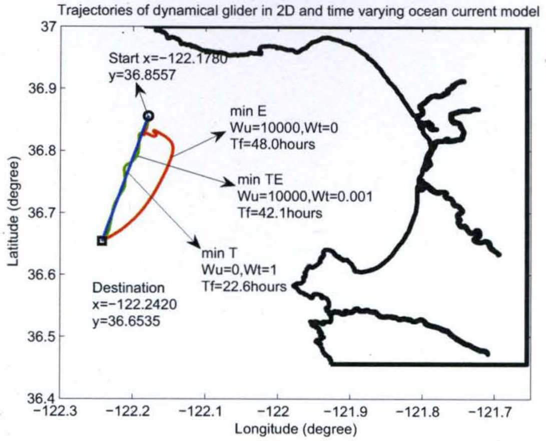


FIGURE 13 – Trajectories of a dynamical glider in the 3D ocean current model.

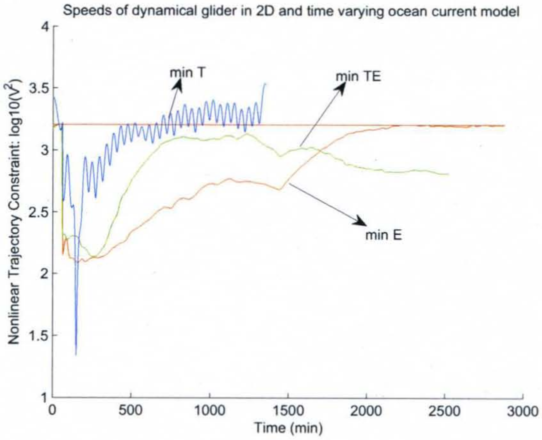


FIGURE 14 – Speed of the dynamical glider in the 3D ocean current model.

3. NTG Solutions for the Dynamic Glider

After applying the dynamic glider model(14) and 3D B-spline ocean current models(8) in NTG, the trajectories of the dynamic glider are plotted in Figure 13 and Figure 14 shows the velocity constraints of the glider. The properties of the trajectories from the dynamic glider model are listed in the following TABLE 3, *3D (Dyn)* represents the trajectories obtained from the 3D B-spline ocean current models and from the dynamic glider model. *Tf*, *Time* and *Energy Cost* represents the same as in TABLE 1, see Section IV-D.

TABLE 3
A DYNAMICAL GLIDER IN THE 3D B-SPLINE OCEAN CURRENT MODEL

$3D(Dyn)$	T_f(hrs)	$Time(s)$	$Energy\,Cost(cm^2/s)$
$min\,E$	48.00	17.87	$5.642e3$
$min\,TE$	42.12	17.60	$6.9491e3$
$min\,T$	22.60	0.95	$1.1389e4$

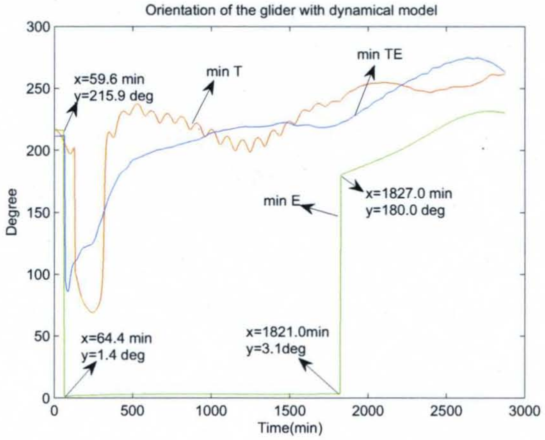


FIGURE 15 – Orientation of the dynamical glider in the 3D ocean current model.

The energy cost is calculated as

$$e = \int_0^T \left((\ddot{y} - \dot{v})^2 + (\ddot{x} - \dot{u})^2 \right) dt \tag{19}$$

The orientation of the glider, shown in Figure 15, is obtained using the following (II.F.3).

$$\tan\theta = \frac{\dot{y} - v}{\dot{x} - u} \tag{20}$$

The figure of the orientation:

The two sharp orientation changes shown in Figure 15 do not violate the constraint given in (17). Specifically, the sharp orientation turn one on the left of green trajectory is

$$\begin{aligned}\frac{d\theta}{dt} &= (1.406 - 215.9)/(64.36 - 59.6)/60 \\ &= -0.75 deg/s \geq -18 deg/s \end{aligned} \tag{21}$$

The sharp turn on the right of the green trajectory is

$$\begin{aligned}\frac{d\theta}{dt} &= (180 - 3.138)/(1827 - 1821)/60 \\ &= -0.49 deg/s \leq 18 deg/s \end{aligned} \tag{22}$$

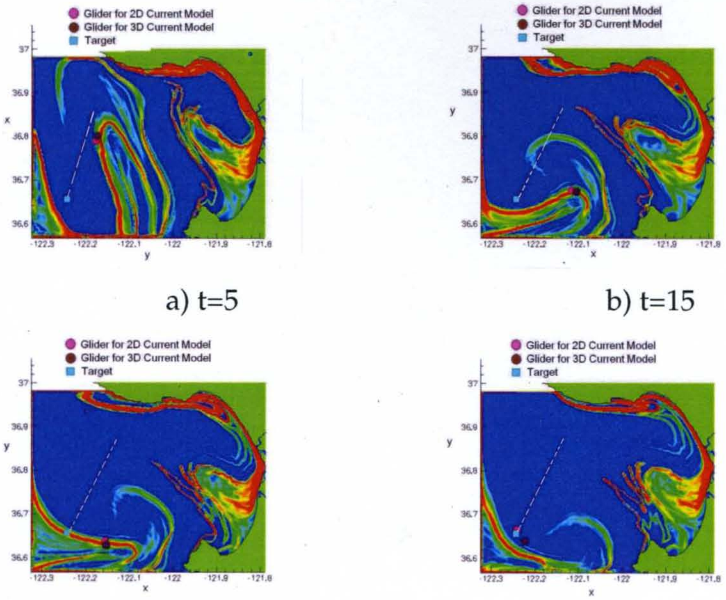
Therefore, the trajectory is satisfied with the constraints about the glider orientation change.

G. Animation of Glider and Ocean Current

The animation of the glider and ocean current is obtained through Tecplot and the results are shown in Figure II.G and Figure II.G for the kinematical and dynamic glider models, respectively. *These new results strengthen our previous hypothesis [11] that LCS in the ocean reveal efficient or near-optimal routes for glider transport.* In Figure II.G and Figure II.G, we have superimposed instances of the min E trajectories given in Figure 11 and Figure 13 with the corresponding LCS fields at that time, respectively. These figures should be thought of as snapshots of a movie which shows the progression of the LCS and the progression of the glider path together. One can see that there is indeed a good correspondence between the optimal trajectory and the LCS.

H. Summary

In this chapter, as an extension to the previous work [11], the ocean current flows 3D B-spline models are established incorporating the time explicitly. These models are applied in the NTG to find the optimal glider trajectories and the results were compared with the previous 2D B-spline models. The results show that



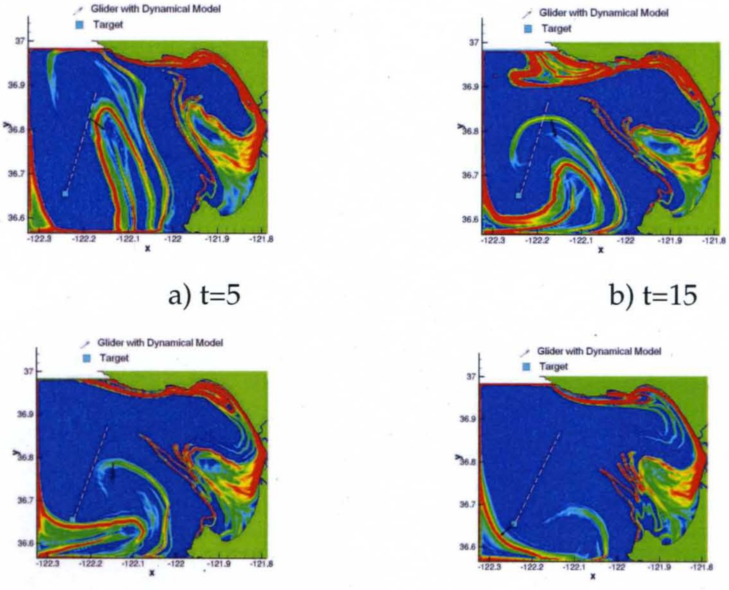
a) t=5

b) t=15

c) t=30

d) t=45

FIGURE 16 – The figure shows the correspondence with the optimal trajectories shown in Figure 11 and an LCS. Note that the red and pink in the figures near the LCS represents the location of the AUVs while the blue represents the final target location.



a) t=5

b) t=15

c) t=30

d) t=48

FIGURE 17 – The figure shows the correspondence with the optimal trajectories shown in Figure 13 and an LCS. Note that the red and pink in the figures near the LCS represents the location of the AUVs while the blue represents the final target location.

the 3D ocean current model is much accurate than 2D ocean current model for the optimal trajectory generation in the ocean currents. The 3D ocean model has produced trajectories with less energy cost. It also eliminated the tedious work to update the current information every one hour as in the 2D ocean model. Hence, it reduced significantly the computational time of obtaining optimal trajectories. Next, the dynamics of the glider is considered in the glider model. The gyroscopic force is applied to control the glider orientation. The new dynamic glider model is used with the 3D B-spline ocean models to produce better trajectories. Finally, Tecplot is used to make the animation movies of the glider traveling in the ocean current. The results enhance our previous hypothesis showing that the trajectory of minimizing energy is reasonably consistent with the LCS.

CHAPTER III

AEROBOT TRAJECTORY GENERATION WITH NTG

A. NASA-JPL Aerobot

The thrill of the unknowns makes people eager to explore the outside far beyond our own planet. The main drawback of the current ground-based robotic planetary vehicles, such as Mars exploration rovers, is their limited range. The 2006 Solar System Exploration Roadmap (SSE) [54] by the National Academy of Sciences indicate that aerial platforms will be required to explore Mars, Venus and Titan shown in Figure 18.

Several types of aerial vehicles such as airplanes, gliders, helicopters, balloons and airships [5] [55] [56] [35] have been considered for aerial robotic planetary exploration. Airplanes and helicopters require significant energy to just stay airborne, flight time of gliders depend mainly on wind, while balloons have limited navigation capabilities. Lighter-Than-Air (LTA) vehicles combine long term mission capabilities and low energy requirements of balloons with flexible maneuverabilities of airplanes. LTA systems, a.k.a. Aerobots or Robotic Airships, bring a new opportunity for the robotic exploration of planets and their moons with atmosphere, such as Mars, Titan and Venus. LTA vehicles have capabilities to travel long distances with limited energy and bring a relatively more in-situ laboratory facilities. They can transport scientific equipments, accomplish regional surveys and wide-area surface mappings. Aerobots can also provide, due to their controllability, precise flight path executions for surveying, station-keeping for extended monitoring high-value science sites, long-range as well as near surface observations, and transportation of scientific equipments. They are also able to execute extensive surveys over solid as well as liquid-covered terrains. Aerobots can reach essentially any point of the planet over multi-month time scales with minimal consumption of limited onboard energy sources. Aerobots can further expand their range by generating opportunistic trajectories making use of winds in planets and moons with atmosphere [57].

The NASA-JPL Aerobot program aims to develop autonomous robotic airships to explore planets and moons with atmosphere, such as Mars, Titan and Venus. They have high potential to overcome the current limitations of the ground-based rovers: limited range. Aerobots or air-based rovers can travel long distances with less energy. Another purpose of designing Aerobots is to allow the robotic air vehicle to travel over rocks instead of around them and hence increasing the versatility, speed and range of the rovers. For instance, seven dark spots near Mars equator have recently been discovered by a Mars-orbiting satellite. They could be entrances to underground Martian caves. The possible caves are called the seven sisters –Dena, Chloe, Wendy, Annie, Abbey, Nikki and Jeanne. Their openings range from about 330 to 820 feet wide. Some researchers have suggested to look into caves for signs of alien life on Mars where there is significant evidence of potential underground aquifers that could support basic, microbial organisms. Robotic air-based rovers might have the advantage of flying over difficult terrain to enter the caverns and explore them whereas land rovers might be cumbersome to do. The NASA-JPL Aerobot program develops a prototype outdoor test-bed and a physically accurate simulation system for testing purposes [58] [59]. The Aerobot is based on an Airspeed Airship AS-800B as shown in Figure 19. The airship specifications are: $11\ m$ in length, $2.5\ m$ in diameter, total volume of $34\ m^3$, two $2.3\ kW(3\ hp)$ and $23\ cm^3(1.4\ cu\ inch)$ fuel engines, double catenary gondola suspension, max. speed of $13\ m/s(25\ kts)$, max. altitude of $500m$, static lift payload of $10\ kg$, and dynamic lift payload of up to $16kg$. The avionics and communication systems are installed in the gondola. It has several onboard sensors such as an IMU (angular rates, linear accelerations), a compass/inclinometer (yaw, roll

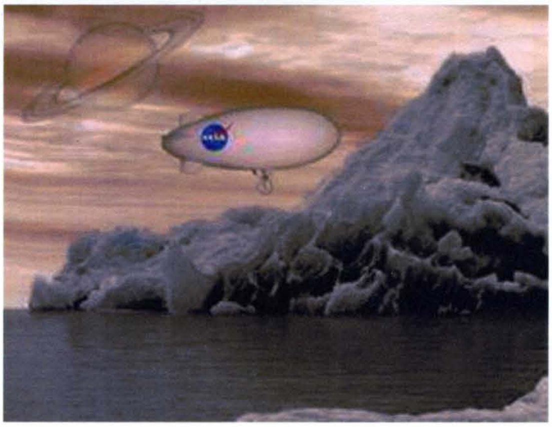


FIGURE 18 – An artists view of an Aerobot exploring a planet [5].

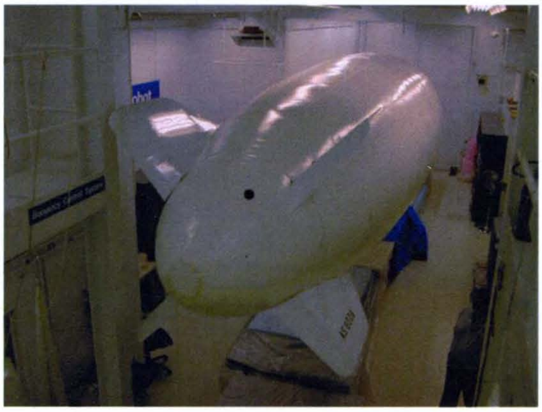


FIGURE 19 – A JPL Aerobot for exploration of Titan and Venus [5].

and pitch angles), laser altimeter (surface relative altitude), barometric altimeter (absolute altitude against reference point), GPS (absolute 3D position), ultrasonic anemometer (3D wind speed) [60], two down looking navigation cameras and a science camera mounted on a pan and tilt unit. The ground station includes a laptop, a graphical user interface to the vehicle, wireless data and video links, video monitors and VCRs, and a differential GPS (DGPS) base station providing differential corrections to the onboard GPS receiver to achieve centimeter accuracy of the 3D position estimates of the vehicle.

Aerobots have different flight modes: take-off, landing, station-keeping, hovering, ascent, descent, high-speed cruise, low-speed flight. These require alternative control strategies and trajectory generation algorithms. Important flight control challenges are non-minimum phase behavior, oscillatory modes at low speeds, time-varying behavior due to altitude variations, and unknown wind disturbances. Even though Aerobots consume modest power, any planetary exploration will require careful management of onboard power sources. Planetary exploration activities such as scientific data gathering, navigation for science site investigation, surface sampling, communications with Earth and/or with an orbiter, control and navigation of the Aerobot, they all require energy. Therefore, Aerobots must use all possible external energy sources. For some planets such as Titan, the Sun is blocked by Titans higher atmosphere. Wind energy for planets and moons with an atmosphere is a very viable source of energy. Therefore, opportunistic trajectory generation algorithms which utilize wind patterns to travel to desired locations are in need to be developed [57]. The wind profile of the atmosphere of some planets such as Mars is known to some degree through observations of previous space missions and atmospheric modeling. The NASA-JPL Aerobot has also an ultrasonic anemometer. This sensor providing estimates of the 3D relative airspeed vector of the Aerobot is used to experimentally obtain the wind profiles. With the specified wind profile, NTG can generate the sub-optimal trajectories for the Aerobot. The objective of this chapter is to guide the Aerobot move by taking

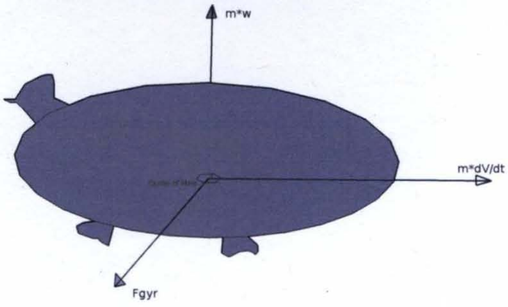


FIGURE 20 – The Euler-Lagrange based Aerobot controls.

advantage of the specified wind to save the energy.

B. Euler-Lagrange Based Aerobot Trajectories

In this work toward obtaining opportunistic trajectory generation for the JPL Aerobot, the Aerobot is modeled by considering its aerodynamics and assuming control inputs are three propellers mounted in the Aerobot which are on the local Cartesian axes. The gyroscopic forces control the velocity in x and y directions, the vertical force control the vertical velocity A dynamical Aerobot shown in Figure 19 modeled as (23) is moving from the point $q_1 = 0, q_2 = 0, q_3 = 0$ to $q_1 = 200, q_2 = 200, q_3 = 200$, which are the coordinates in the Cartesian system. Still, the following assumptions are made:.

- the Aerobot center of gravity is at the same location as the center of buoyancy.
- the Aerobot roll rotation is small enough to disregard, the pitch, yaw angles of Aerobot are automatically consistent with their velocity directions.
- the reference system is Cartesian, the shape of earth is disregarded.
- the temperature is kept as constant and the air flow is un-compressible.

The Aerobot has three controlled inputs. The horizontal orientation is controlled by gyroscopic forces (23), the ascending and descending velocities are controlled by a force $m * w$,and m is the mass of the Aerobot. The mass of the Aerobot is simplified as 1. For consideration of rotational energy, the rotational inertial I is also regarded as 1.

$$\begin{aligned}
\dot{q}_1 &= V\cos\phi\cos\theta + u;\\
\dot{q}_2 &= V\cos\phi\sin\theta + v;\\
\dot{q}_3 &= V\sin\phi
\end{aligned}$$

where V is the forward velocity of the Aerobot. ϕ is the pitch angle, θ is the yaw angle.

$$\begin{aligned}
&\ddot{q}_1 = -V\sin\phi\cos\theta\tfrac{d\phi}{dt} - V\cos\phi\sin\theta\tfrac{d\theta}{dt} + \dot{u};\\
&\ddot{q}_2 = -V\sin\phi\sin\theta\tfrac{d\phi}{dt} + V\cos\phi\cos\theta\tfrac{d\theta}{dt} + \dot{v};\\
&\ddot{q}_3 = V\cos\phi\tfrac{d\phi}{dt};
\end{aligned}$$

$$F_{gyr} = \begin{pmatrix} -\tau\left(\sin\phi\cos\theta\dfrac{d\phi}{dt} + \cos\phi\sin\theta\dfrac{d\theta}{dt}\right) = f_1gyr \\ -\tau\left(\sin\phi\cos\theta\dfrac{d\phi}{dt} - \cos\phi\cos\theta\dfrac{d\theta}{dt}\right) = f_2gyr \\ \tau\left(V\cos\phi\dfrac{d\phi}{dt}\right) = f_3gyr \end{pmatrix} \tag{23}$$

Where q_1, q_2, q_3 represents x, y and z. u, v are the wind velocities in the x and y directions, τ is the control input for the gyroscopic force. The Aerobot roll, pitch, yaw angles are correspondingly represented as ψ, ϕ, θ, the respective angular velocities are denoted as $\dot{\psi} = p, \dot{\phi} = q, \dot{\theta} = r$.

1. Euler-Lagrange Equations

The optimal control problem for NTG is to obtain $f(t)$ to minimize the cost function.

$$J(q, f) = \int_{t_0}^{t_f} C(q(t), \dot{q}(t), f(t))dt \tag{24}$$

At the same time, the motion of $q(t)$ of the mechanic system from $(q^{t_0}, \dot{q}^{t_f})$ to a state $(q^{t_0}, \dot{q}^{t_f})$ is to satisfy the Lagrange-d'Alembert principle, which requires that (25).

$$\delta\int_{t_0}^{t_f} L(q(t), \dot{q}(t))dt + \int_{t_0}^{t_f} f(t)\cdot\delta q(t)dt = 0 \tag{25}$$

It can be expressed as

$$\int_{t_0}^{t_f} (\frac{\partial L}{\partial q}\delta q + \frac{\partial L}{\partial \dot{q}}\delta \dot{q})dt + \int_{t_0}^{t_f} f(t)\delta q(t)dt = 0 \tag{26}$$

$\Longrightarrow$

$$\int_{t_0}^{t_f} (\frac{\partial L}{\partial q}\delta q dt + \int_{t_0}^{t_f} \frac{\partial L}{\partial \dot{q}} d\delta q) + \int_{t_0}^{t_f} f(t)\delta q(t)dt = 0 \tag{27}$$

Because of:

$$\int_{t_0}^{t_f} \frac{\partial L}{\partial \dot{q}} d\delta q = \frac{\partial L}{\partial \dot{q}}\delta q_{t_0}^{t_f} - \int_{t_0}^{t_f} \delta q \cdot \frac{d}{dt}\frac{\partial L}{\partial \dot{q}} \cdot dt + \int_{t_0}^{t_f} f(t)\delta q(t)dt = 0 \tag{28}$$

For variations $\delta q(t_0) = \delta q(t_f) = 0$, thus

$$\int_{t_0}^{t_f} \frac{\partial L}{\partial q}\delta q dt - \frac{d}{dt}\frac{\partial L}{\partial \dot{q}}\delta q dt + f(t)\delta q(t)dt = 0 \tag{29}$$

Finally, the continuous Euler-Lagrange equation

$$\frac{\partial L}{\partial q} - \frac{d}{dt}\frac{\partial L}{\partial \dot{q}} + f(t) = 0 \tag{30}$$

Since the Aerobot is controlled by gyroscopic forces in the horizontal plane and a vertical force along the z axes, the Euler-Lagrange equations should be satisfied along the x, y and z axes in the local reference frame. The Lagrange L of the system is the kinetic energy KE minus the potential energy PE.

$$L = KE - PE \tag{31}$$

The kinetic energy of the Aerobot should be expressed as the sum of the translational kinetic energy of the center of mass and the rotational kinetic energy about the center of mass. For a given fixed axis of rotation, the kinetic energy can be expressed in the form of

$$KE = KE_{rotation} + KE_{translation} = \frac{1}{2}Iw^2 + \frac{1}{2}mv^2 \tag{32}$$

For the local reference system, the Euler-Lagrange equations along x, y, z axes are presented as

$$\begin{aligned} &\frac{\partial L_x}{\partial x} - \frac{d}{dt}\frac{\partial L}{\partial \dot{x}} + f_x = 0; \\ &\frac{\partial L_y}{\partial y} - \frac{d}{dt}\frac{\partial L}{\partial \dot{y}} + f_y = 0; \\ &\frac{\partial L_z}{\partial z} - \frac{d}{dt}\frac{\partial L}{\partial \dot{z}} + f_z = 0; \end{aligned} \tag{33}$$

And the Lagranges of the Aerobot are listed as:

$$\begin{aligned} L_x &= \frac{1}{2}I_x p^2 + \frac{1}{2}mv_x^2; \\ L_y &= \frac{1}{2}I_y q^2 + \frac{1}{2}mv_y^2; \\ L_z &= \frac{1}{2}I_z r^2 + \frac{1}{2}mv_z^2 - mgz; \end{aligned} \tag{34}$$

where mgz is the potential energy of the Aerobot. g is the gravity acceleration. The control forces f_x, f_y, f_z are the summation of all effective forces along the three axes.

$$\begin{aligned} f_x &= f_1 gyr + f_{xd}; \\ f_y &= f_2 gyr + f_{yd}; \\ f_z &= f_3 gyr + f_{zl} + f_{bl}; \end{aligned} \tag{35}$$

where $f_1 gyr, f_2 gyr$ are the gyroscopic forces in the horizontal plane. $f_3 gyr$ is the control force to move the Aerobot vertically. f_{xd}, f_{yd} reflect the drag force due to the aerodynamics while f_{zl} is the lift force. f_{bl} is the buoyancy force due to the Helium in the Aerobot. Due to velocity up-limit of the Aerobot, the air or wind in the field is considered as a piece of uncompressed and inviscid flow. Therefore, the Euler-Lagrange equations 33 are transformed into

$$\begin{aligned} -m\left(\ddot{x} - \dot{u}\right) + f_1 gyr + f_{xd} &= 0; \\ -m\left(\ddot{y} - \dot{v}\right) + f_2 gyr + f_{yd} &= 0; \\ -mg - m\ddot{z} + f_3 gyr + f_{zl} + f_{bl} &= 0; \end{aligned} \tag{36}$$

$$\begin{aligned} v_x &= V\cos\phi\cos\theta = \dot{x} - u; \\ v_y &= V\cos\phi\sin\theta = \dot{y} - v; \\ v_z &= V\sin\phi = \dot{z} \end{aligned} \tag{37}$$

V is the forward velocity, ψ, θ are the yaw and pitch angles of the Aerobot respectively. For aerodynamics, the drag and lift forces of the Aerobot are derived due to the flows around the Aerobot when it is flying at certain speeds [61] [62].

$$\begin{aligned} f_{xd} &= C_{xd} A \rho_{air} V^2 \cos\psi \cos\theta; \\ f_{yd} &= C_{yd} A \rho_{air} V^2 \sin\psi \sin\theta; \\ f_{zl} &= C_{zl} A \rho_{air} V^2 \sin\psi; \\ f_{bl} &= (\rho_{air} - \rho_{helium}) \Gamma g; \end{aligned} \tag{38}$$

where ρ_{air} is the mean density difference of ambient air. ρ_{helium} is the mean density of helium in the Aerobot envelope. ψ is the yaw angle of the Aerobot. C_{xd}, C_{yd}, C_{zl} are the coefficients of the drag and lift forces which are dependent on the physical parameters of the Aerobot such as the volume, the shape, the pitch angle and the material frictions of the Aerobot surface. A is the reference area. It is chosen as $(Buoyant\ Volume)^{2/3}$ [62]. Γ is the volume of the Aerobot, g is the constant gravity acceleration on earth [63].

2. Wind Profile

The wind profile is modified from the paper [57], the research area is restricted in the cube which the reference point is $(0, 0, 0)$ to $(200, 200, 200)$. Assuming the wind profile is layered horizontally, no upward or downward wind exists. The wind velocity vectors at each layer are considered as known.

$$(u, v) = \begin{cases} (10, 10), & \text{for } q_3 \subseteq (0, 50) \\ (-10, 10), \text{for } q_3 \subseteq (50, 100) \\ (10, -10), \text{for } q_3 \subseteq (100, 150) \\ (0, -10), & \text{for } q_3 \subseteq (150, 300) \end{cases} \tag{39}$$

where q_3 is the coordinate in the vertical direction. In Figure 21, the x, y, z are the coordinates of the system, respectively represent q_1, q_2, q_3.

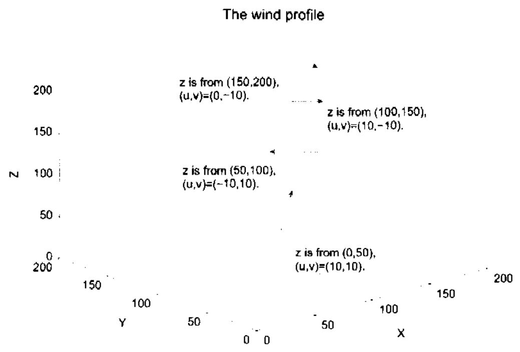


FIGURE 21 – The wind profile for the state-space based Aerobot.

3. Problem Formulation

The cost function and constraints are listed in the following. The cost function J is

$$J = W_t(t_f - t_0) + W_u \int_{t_0}^{t_f} \| F_{gyr} \|^2 \, dt + W_v V^2 \tag{40}$$

where W_t, W_u, W_v, W_q are the weights, t_f is the unknown final time for the trajectory. The constraints:

- (Linear) Initial Constraints:

$$0 - \epsilon \leq q_1(t_0) \leq 0 + \epsilon$$

$$0 - \epsilon \leq q_2(t_0) \leq 0 + \epsilon$$

$$0 - \epsilon \leq q_3(t_0) \leq 0 + \epsilon$$

$$0 \leq t_f - t_0 \leq 100 \ s$$

- (Linear) Final Constraints:

$$200 - \epsilon \leq q_1(t_f) \leq 200 + \epsilon$$

$$200 - \epsilon \leq q_2(t_f) \leq 200 + \epsilon$$

$$200 - \epsilon \leq q_3(t_f) \leq 200 + \epsilon$$

- (Nonlinear) Trajectory Constraints:

$$0 - \epsilon \leq q_1(t) \leq 300 + \epsilon$$

$$0 - \epsilon \leq q_2(t) \leq 300 + \epsilon$$

$$0 - \epsilon \leq q_3(t) \leq 300 + \epsilon$$

$$0 - \epsilon \leq V \leq 400 + \epsilon$$

where $q_1(t_0)$, $q_2(t_0)$, $q_3(t_0)$, $q_1(t_f)$, $q_2(t_f)$, $q_3(t_f)$ are the initial and final location of the Aerobot. $q_1(t)$, $q_2(t)$ and $q_3(t)$ are the positions of the Aerobot in the trajectory. V is the horizontal forward velocity of Aerobot. ϵ is a small number.

Since the Aerobot is controlled by the gyroscopic force and the vertical propeller. The trajectory is satisfied with Euler-Lagrange equations. According to (36), (37) and (38), the Euler-Lagrange equations are expressed as:

$$\begin{aligned} -m\left(\ddot{x} - \dot{u}\right) + f_1 gyr + C_{xd}\rho_{air}V^2\cos\psi &= 0; \\ -m\left(\ddot{y} - \dot{v}\right) + f_2 gyr + C_{yd}\rho_{air}V^2\sin\psi &= 0; \\ -mg - m\ddot{z} + f_3 gyr + C_{zl}\rho_{air}V^2 + (\rho_{air} - \rho_{helium})\Gamma g &= 0; \end{aligned} \tag{41}$$

.

.

4. Simulated 3D Trajectory

The prototype Aerobot [64] testbed developed at JPL is based on an Airspeed Airship AS-800 B (Figure 19). The parameters for the Aerobot are : length of $11\ m$, diameter of $2.5\ m$, total volume of $34\ m^3$, two 2.3 kW (3 hp) 23 cm^3 fuel engines, double catenary gondola suspension. Assuming maximum speed of the Aerobot is $20\ m/s$, maximum ceiling of $500\ m$.

When the Aerobot is modeled as (23), the physical parameters are simplified with lift and drag coefficients are both set to be 1. The buoyancy is considered to be zero with the mass is considered to be 1. The wind profile is assumed to be known

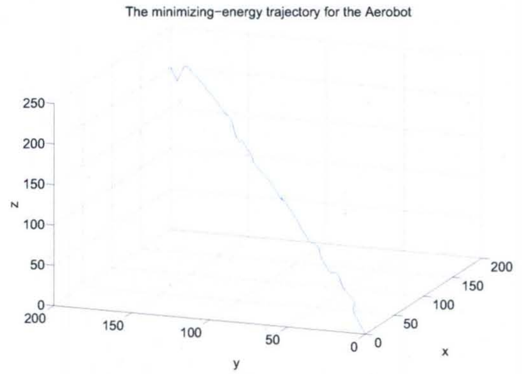


FIGURE 22 – The minimizing-energy trajectory generated by NTG for the Aerobot.

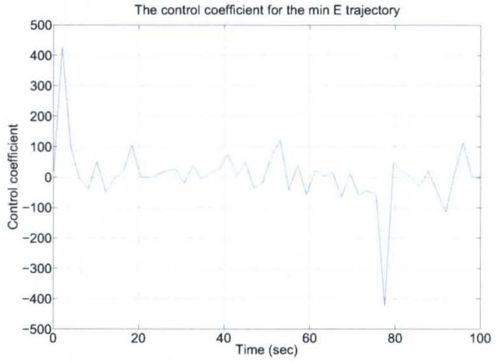


FIGURE 23 – The control input τ for the minimizing-energy trajectory.

as 39 and, NTG generated the following 3D trajectory presented in Figure 22. The energy cost for this trajectory is $4.1659e7$, the final time is 100 second, the running time is less than 29.12 seconds. From this example, we have shown NTG can generate the reasonable optimal trajectory with the specified wind profile. The control input τ as in (23) The minimizing-energy trajectory shown in Figure 22, the trajectory is taking advantage of the wind profile to save energy. The roll angle is zero, while the yaw angle is always point to the destination, which is 45 degrees. NTG tried to generate the minimizing-time trajectory, the final time is 87.35 second. The running time is 11.46 seconds The energy cost is $12.584e7$. The control input τ for the min T trajectory is presented as: For the minimizing time trajectory, it is almost a straight line, the Aerobot roll angle is zero, while the yaw angle is always point to the destination, which is 45 degrees. The computer specifica-

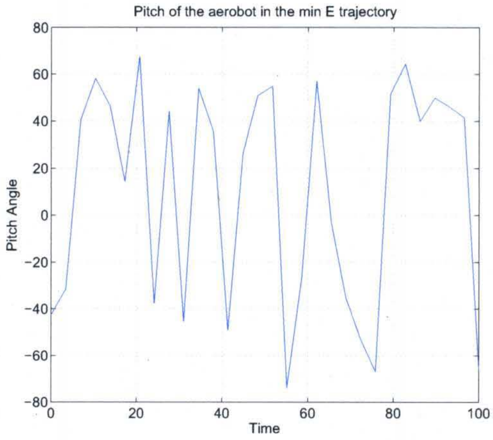


FIGURE 24 – The pitch angle ϕ for the minimizing-energy trajectory.

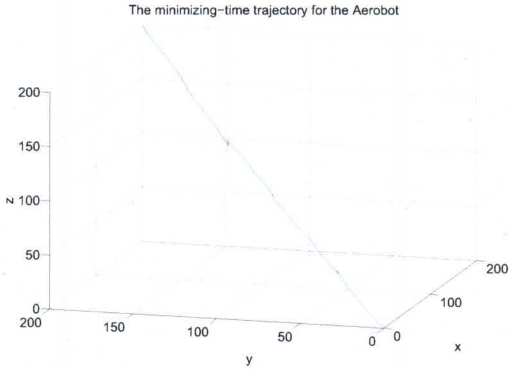


FIGURE 25 – The minimizing-time trajectory generated by NTG for the Aerobot.

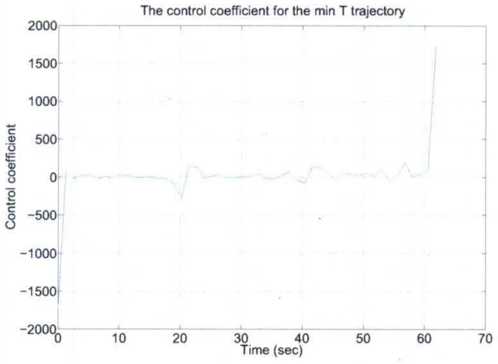


FIGURE 26 – The control input τ for the minimizing-time trajectory.

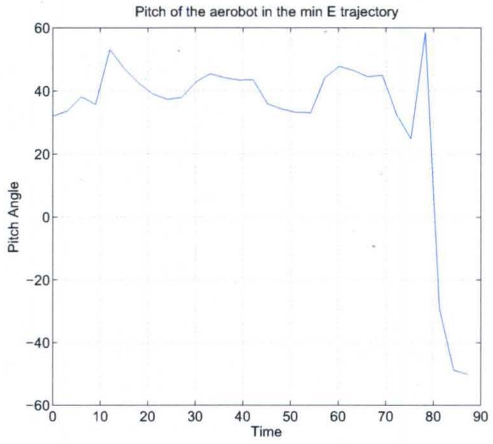


FIGURE 27 – The pitch angle ϕ for the minimizing-time trajectory.

TABLE 4
3D TRAJECTORIES GENERATED BY NTG FOR THE EULER-LAGRANGE BASED AEROBOT

NTG	$T_f(s)$	$Time(s)$	$Energy\,Cost(m^2/s)$
$min\,E$	100.00	29.12	$8.336e7$
$min\,T$	87.35	11.46	$14.466e7$

tion of the simulation is Ubuntu 7.10, Kernel Linux 2.6.22-14-386, Memory 2.0 GB, AMD Athlon(tm) 64×2 Dual Core Processor 3800+. TABLE 4 shows the trajectories generated by NTG for the modeled Aerobot are reasonable considering the minimizing-time trajectory is the straight line and the energy cost is larger than the minimizing-energy trajectory. These two trajectories are presented in Figure 22 and Figure 25, respectively.

C. State Space Model Based Trajectories

When the Aerobot is modeled as a state-space model, the trajectories can be generated with more time. It is reasonable to assume with ordinary computation capability, trajectories can only be generated off line with all other conditions known in advance. For this demonstration, the decoupled longitudinal and lateral

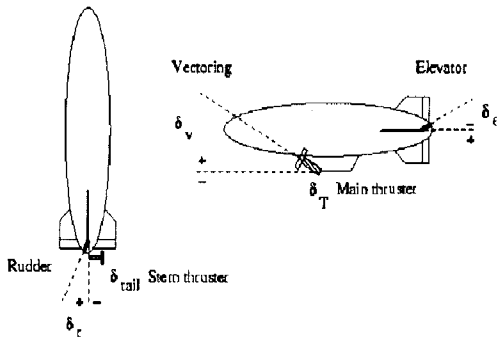


FIGURE 28 – The state-space based Aerobot controls [6]

equations of motion of the Aerobot are from AURORA (Autonomous Unmanned Remote Monitoring Robotic Airship) project [65].

The state-space model is decoupled into longitudinal and lateral motions. The control inputs as elevator deflection δ_e, thrust demand δ_T, vectoring angle δ_v for the longitudinal motion, and aileron deflection δ_a, rudder deflection δ_r for the lateral motion. The outputs are the velocities and orientation of the airship. The airship control inputs and their positive references are shown in Figure 28 The airship is moving from the point $q_1 = 0, q_2 = 0, q_3 = 0$ to $q_1 = 200, q_2 = 200, q_3 = 200$, which are the coordinates in the Cartesian system. Still, the following assumptions are made:. The linearized state-space model is obtained from nonlinear dynamic equation of the airship given by [6], resulting into decoupled longitudinal and lateral motions. For the longitudinal motion, the output vector is

$$X_v(t) = [u, w, q, \theta] \tag{42}$$

where u is the longitudinal component of the airship absolute speed which is relative to the air, w its vertical component, q is the pitch rate and θ is the pitch angle. The control vector for the longitudinal motion is

$$U_v(t) = [\delta_e, \delta_T, \delta_v] \tag{43}$$

where δ_e is the elevator deflection, δ_T is the thrust demand and δ_v is the vectoring angle. The equation of longitudinal motion is listed as

$$\dot{X}_v = A_v X_v(t) + B_v U_v(t) \tag{44}$$

where A_v and B_v are numerically linearized system matrices [6] as

$$A_v = \begin{bmatrix} -0.1569 & -0.0651 & 1.8059 & -0.4522 \\ -0.0965 & -0.6300 & 8.0737 & -1.3584 \\ 0.0178 & -0.1059 & -3.7053 & -0.8279 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$B_v = \begin{bmatrix} 1.83876 & 0.0531 \\ -1.5921 & -0.0003 \\ -1.1832 & 0.0074 \\ 0 & 0 \end{bmatrix}$$

For its lateral motion, the output vector is

$$X_h(t) = [v, p, r, \theta] \tag{45}$$

where v is the lateral component of the airship absolute velocity, p and r are the roll and yaw rates, θ is the roll angle. The control vector is given by

$$U_h(t) = [\delta_a, \delta_r] \tag{46}$$

where δ_a is the aileron deflection, δ_r is the rudder deflection. Its lateral motion of equation is presented as

$$\dot{X}_h = A_h X_h + B_h U_h \tag{47}$$

where A_h and B_h are numerically linearized matrices from [66] as

$$A_h = \begin{bmatrix} 0.0378 & 0.4037 & 1.8059 & -2.5864 \\ 1.5641 & -0.6429 & 8.0737 & -6.3747 \\ -0.4161 & -1.4674 & -6.2235 & -0.0225 \\ 0 & 1 & 0.0913 & 0 \end{bmatrix}$$

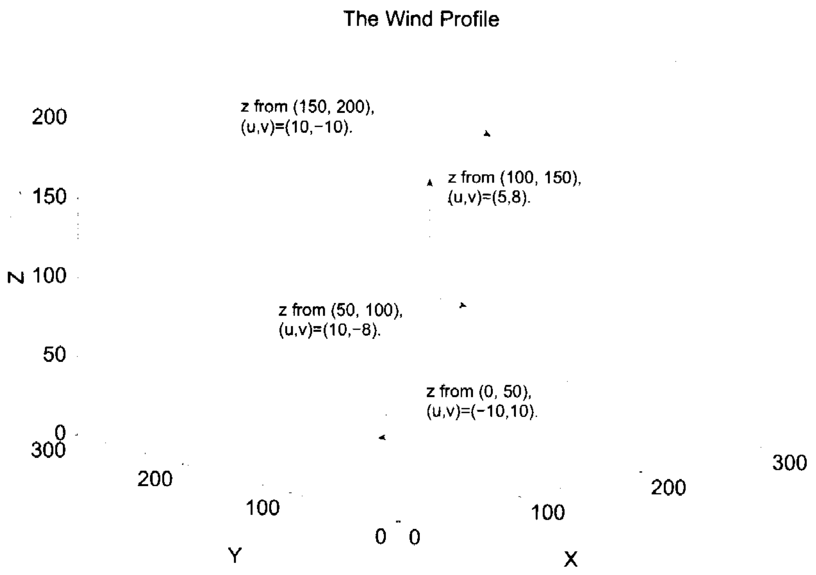


FIGURE 29 – The wind profile for the state-space based Aerobot.

$$B_h = \begin{bmatrix} -7.1360 & 4.5273 \\ -13.4035 & 3.07573 \\ -0.2389 & -2.9211 \\ 0 & 0 \end{bmatrix}$$

For the state-space model, the wind profile is modeled as (48), the research area is restricted in the cube which the reference point is $(0, 0, 0)$ to $(200, 200, 200)$ meters. Assuming the wind profile is layered horizontally, no upward or downward wind exists. The wind velocity vectors at each layer are considered as known.

$$(u, v) = \begin{cases} (-10, 10), \text{for } q_3 \subseteq (0, 50) \\ (10, -8), \quad \text{for } q_3 \subseteq (50, 100) \\ (5, 8), \qquad \text{for } q_3 \subseteq (100, 150) \\ (10, -10), \text{for } q_3 \subseteq (150, 300) \end{cases} \tag{48}$$

where q_3 is the coordinate in the vertical direction. In Figure 29, the x, y, z are the coordinates of the system, respectively represent q_1, q_2, q_3.

1. Problem Formulation

The cost function and constraints are listed in the following. The cost function J is

$$J = W_t(t_f - t_0) + W_u \int_{t_0}^{t_f} ((\frac{\dot{x}}{dt} - windu)^2 + (\frac{\dot{y}}{dt} - windv)^2 + (\frac{\dot{z}}{dt})^2)dt \tag{49}$$

where W_t, W_u are the weights. For minimizing time trajectory, W_t is equal to 1000, while W_u are both 0. For minimizing energy trajectory, W_t is set to be 0, while W_u are both set to be 10. t_f is the unknown final time for the trajectory. The constraints:

- (Linear) Initial Constraints:

$$0 - \epsilon \leq q_1(t_0) \leq 0 + \epsilon$$

$$0 - \epsilon \leq q_2(t_0) \leq 0 + \epsilon$$

$$0 - \epsilon \leq q_3(t_0) \leq 0 + \epsilon$$

$$0 \leq t_f - t_0 \leq 200\ s$$

- (Linear) Final Constraints:

$$200 - \epsilon \leq q_1(t_f) \leq 200 + \epsilon$$

$$200 - \epsilon \leq q_2(t_f) \leq 200 + \epsilon$$

$$200 - \epsilon \leq q_3(t_f) \leq 200 + \epsilon$$

- (Linear) Trajectory Constraints:

$$0 - \epsilon \leq q_1(t) \leq 300 + \epsilon$$

$$0 - \epsilon \leq q_2(t) \leq 300 + \epsilon$$

$$0 - \epsilon \leq q_3(t) \leq 300 + \epsilon$$

- (Linear) Control Inputs Constraints:

$$-1 \leq \delta_e \leq$$

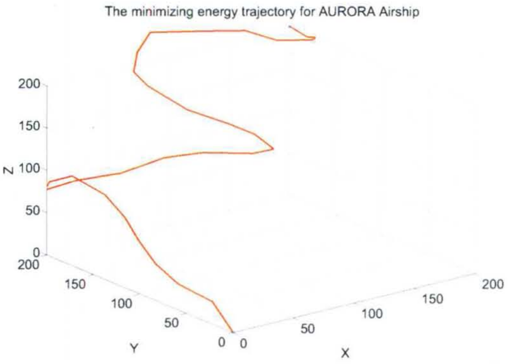


FIGURE 30 – The minimizing-energy trajectory for the state-space based Aerobot.

$$-100 \leq \delta_T \leq 100$$

$$-0.5 \leq \delta_v \leq 0.5$$

$$-1 \leq \delta_a \leq 1$$

$$-1 \leq \delta_r \leq 1$$

where $q_1(t_0)$, $q_2(t_0)$, $q_3(t_0)$, $q_1(t_f)$, $q_2(t_f)$, $q_3(t_f)$ are the initial and final location of the Aerobot. $q_1(t)$, $q_2(t)$ and $q_3(t)$ are the positions of the Aerobot in the trajectory. ϵ is a small number.The other constraints are nonlinear constraints listed as 44 and 47.

2. Simulated 3D Trajectories

When the Aerobot is modeled as (44) and (47), the wind profile is assumed to be known as in (39). NTG generated the minimizing-energy 3D trajectory in Figure 30. The energy cost for this trajectory is $4.2297e3$, the final time is 183.97 seconds, the computation time is about 18 minutes. The longitudinal and lateral constraints make the computation time is as long as 18 minutes, it means that the trajectory has to be obtained by off-line with the available wind profile in advance. The control inputs elevator deflection δ_e, thrust demand δ_T, and aileron deflection δ_a, rudder deflection δ_r are shown in the following.

The vectoring deflection δ_v is not shown here considering that it is not explicitly shown in the longitudinal and lateral dynamics constraints.

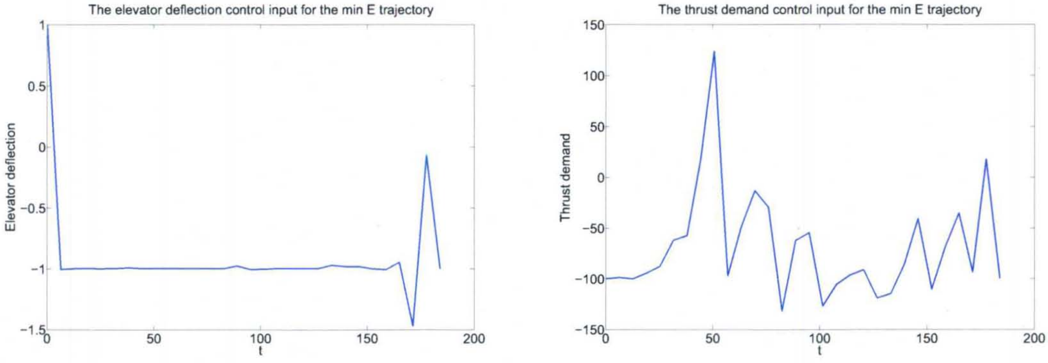


FIGURE 31 – The elevator deflection δ_e and the thrust demand δ_T for Figure 30

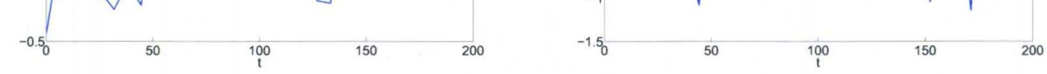


FIGURE 32 – The elevator deflection δ_a and the thrust demand δ_r for Figure 30

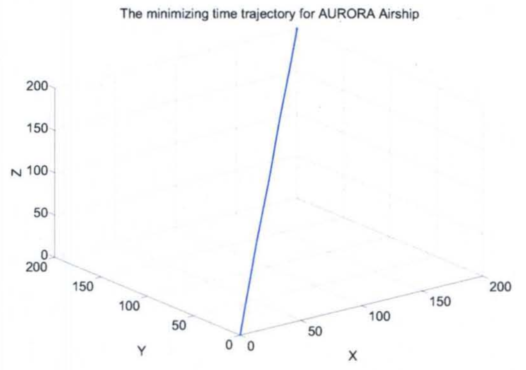


FIGURE 33 – The minimizing-time trajectory for the state-space based Aerobot

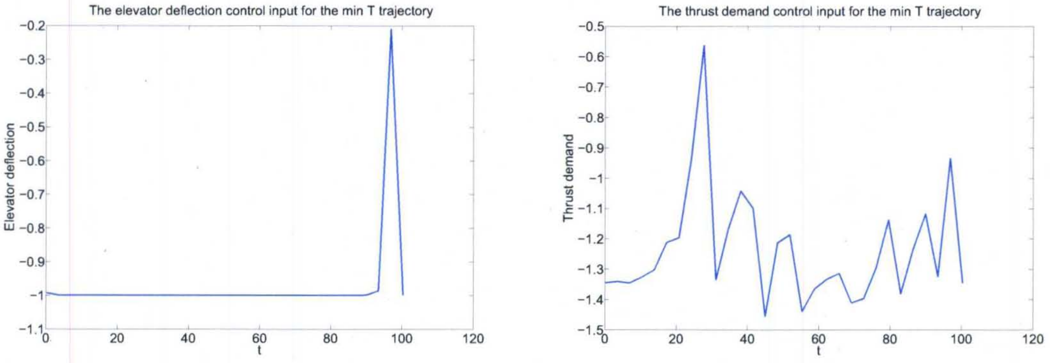


FIGURE 34 – The elevator deflection δ_e and the thrust demand δ_T for Figure 33

When the trajectory is trying to minimize the time, the trajectory is not going with the wind profile. It just go straight to the destination as it is shown in Figure 33. For the minimizing-time trajectory, the final time is 100.30 seconds. The computation time of the NTG algorithm is 6 minutes. The energy cost is $4.6963e3$. The minimizing time trajectory control inputs elevator deflection δ_e, thrust demand δ_T, and aileron deflection δ_a, rudder deflection δ_r are also shown in the following.

The simulation platform is Ubuntu 7.10, Kernel Linux 2.6.22-14-386, Memory 2.0 GB, AMD Athlon(tm) 64×2 Dual Core Processor 3800+.

TABLE 5 shows the trajectories generated by NTG for the modeled Aerobot are reasonable considering the minimizing-time trajectory is the straight line and the energy cost is larger than the minimizing-energy trajectory.

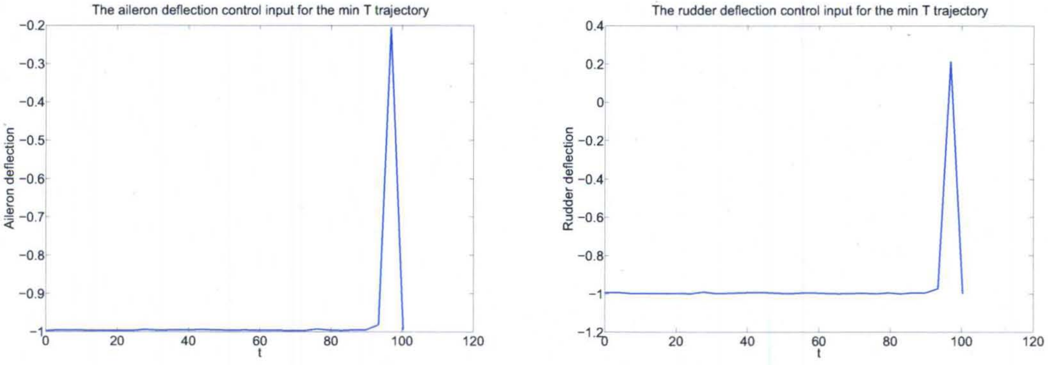


FIGURE 35 – The elevator deflection δ_a and the thrust demand δ_r for Figure 33

TABLE 5
3D TRAJECTORIES GENERATED BY NTG FOR THE STATE-SPACE BASED AEROBOT

NTG	$T_f(s)$	$Time(m)$	$Energy\ Cost(m^2/s)$
$min\ E$	183.97	18	$4.2297e3$
$min\ T$	100.29	5	$4.6963e3$

D. Summary

This chapter shows that the JPL Aerobot energy efficient trajectories can be generated by NTG. This problem is investigated from two perspectives, one is from the energy perspective, another is from the state-space based model. The former one is much faster to calculate than the later one. Both for the Euler-Lagrange constraints based trajectories and the state-space based models NTG can be utilized to generate energy-efficient trajectory for the JPL Aerobot.

CHAPTER IV

TRAJECTORY GENERATION WITH DMOC

A. DMOC Methodology

Based on discrete Lagrangian mechanics [67] [68], DMOC (Discrete Mechanical and Optimal Control) [9] is proposed to solve optimization control problems both for mechanical systems. Its application is mainly in the control of mechanical systems, such as trajectory generation for a glider, control of Compass Gait Biped [27], formation of flying spacecrafts [69]. Also DMOC can be applied in solving variational problems in computer vision and graphics [70], poit vortices [71].

The innovative part of DMOC is to exploit the variational structure directly. Instead of first deriving the Euler-Lagrange equations (equations of motion) for the system, it utilizes a global discretization of the states and the controls by the discrete Lagrange-d'Alembert principle to obtain equality constraints, then the problem is transformed into a finite dimensional nonlinear optimization problem. While in NTG the collocation method [72] is to choose a finite-dimensional space of candidate solutions and a number of points (collocation points) in the domain, and to select a solution which satisfies with the given cost and constraints equations at collocation points.

Considering a mechanical system with configuration space Q is to move on a curve $q(t) \in Q$ in the time period of $[t_0, t_f]$ from a state $(q^{t_0}, \dot{q}^{t_0})$ to a state $(q^{t_f}, \dot{q}^{t_f})$ under a control force $f(t)$, the cost function in this optimal control problem is given as:

$$J(q,f) = \int_{t_0}^{t_f} C(q(t), \dot{q}(t), f(t))dt \tag{50}$$

where $q(t)$ is the state of the system. For optimal control, $f(t)$ is chosen so that the cost function is minimized. The constraints are listed as in (51).

$$\begin{array}{llclcl} \text{Initial} & lb_0 & \leq & \Psi_0(\mathbf{q}(t_0), \mathbf{f}(t_0), t_0) & \leq & ub_0 \\ \text{Trajectory} & lb_t & \leq & \Psi_t(\mathbf{q}(t), \mathbf{f}(t), t) & \leq & ub_t \\ \text{Final} & lb_f & \leq & \Psi_f(\mathbf{q}(t_f), \mathbf{f}(t_f), t_f) & \leq & ub_f \end{array} \tag{51}$$

At the same time, motion $q(t)$ of the system is satisfied with the Lagrange-d'Alembert principle, which requires that (52).

$$\delta \int_{t_0}^{t_f} L(q(t), \dot{q}(t))dt + \int_{t_0}^{t_f} f(t) \cdot \delta q(t)dt = 0 \tag{52}$$

where $L : TQ \to \mathbb{R}$ is the Lagrange of the mechanical system. The variations δq in two terminals are $\delta q(t_0) = \delta q(t_f) = 0$.

For a trajectory generation problem with a cost function as (50), constraint functions as (51). The time for system states is discretized as $0, h, 2h, ...Nh = t_f$, where h is the step size and $N \in \mathbb{N}$. The continuous state $q(t)$ and the continuous force $f(t)$ is approximated by discrete states $q_d(kh)$ and forces $f_d(kh)$.

Through direct discretization, the Lagrangian in Euler-Lagrange equation (52) is approximated over a time slice $[kh, (k+1)h]$ by a discrete Lagrangian L_d.

$$L_d(q_k, q_{k+1}) \approx \int_{kh}^{(k+1)h} L(q(t), \dot{q}(t))dt, \tag{53}$$

At the same time, the virtual work in (52) also can be approximated as

$$f_k^- \cdot \delta q_k + f_k^+ \cdot \delta q_{k+1} \approx \int_{kh}^{(k+1)h} f(t) \cdot \delta q(t)dt, \tag{54}$$

where f_k^-, f_k^+ are called left and right discrete force respectively.

$$f_k^- = f(kh) \cdot h/2; \tag{55}$$

$$f_k^+ = f((k+1)h) \cdot h/2 \tag{56}$$

The discrete version of Lagrange-d'Alembert principle (52) requires that

$$\delta \sum_{k=0}^{N-1} L_d(q_k, q_{k+1}) + \sum_{k=0}^{N-1} (f_k^- \cdot \delta q_k + f_k^+ \cdot \delta q_{k+1}) = 0. \tag{57}$$

It can deduce into the following equation with derivative operations according to variables q_k and q_{k+1} by D_1 and D_2, respectively.

$$\sum_{k=0}^{N-1} D_1 L_d(q_k, q_{k+1}) \cdot \delta q_k + \sum_{k=0}^{N-1} D_2 L_d(q_k, q_{k+1}) \cdot \delta q_{k+1} + \sum_{k=0}^{N-1} f_k^- \cdot \delta q_k + f_k^+ \cdot \delta q_{k+1} = 0. \tag{58}$$

Then, the equation can be transformed into

$$\sum_{k=0}^{N-1} D_1 L_d(q_k, q_{k+1}) \cdot \delta q_k + \sum_{k=1}^{N-1} D_2 L_d(q_{k-1}, q_k) \cdot \delta q_k + \sum_{k=1}^{N-1} f_k^- \cdot \delta q_k + f_{k-1}^+ \cdot \delta q_k = 0. \tag{59}$$

It can be expressed as:

$$\sum_{k=1}^{N-1} (D_1 L_d(q_k, q_{k+1}) + D_2 L_d(q_{k-1}, q_k) + f_{k-1}^+ + f_k^-) \delta q_k = 0. \tag{60}$$

For all variations δq_k, $\delta q_0 = \delta q_N = 0$, the discrete Lagrange-d'Alembert principle is derived as

$$D_1 L_d(q_k, q_{k+1}) + D_2 L_d(q_{k-1}, q_k) + f_{k-1}^+ + f_k^- = 0, \tag{61}$$

where $k = 1 \ldots N - 1$, called the forced discrete Euler-Lagrange equations. Other constraints such as (51) are also discretized with corresponding discrete states and forces. Discrete boundary conditions at $t_0 = 0$ and $t_f = Nh$ are expressed as

$$\begin{aligned} D_2 L(q_0, \dot{q}_0) + D_1 L_d(q_0, q_1) + f_0^- &= 0, \\ -D_2 L(q_N, \dot{q}_N) + D_2 L_d(q_{N-1}, q_N) + f_{N-1}^+ &= 0. \end{aligned} \tag{62}$$

B. DMOC Tutorial

This section presents a detailed procedure [73] to apply DMOC methodology to solve optimal control problems. It explains the principle of DMOC, and how to formulate the problem in DMOC. Then the steps are shown about how to install and configure nonlinear programming solver IPOPT, and how to use the modeling

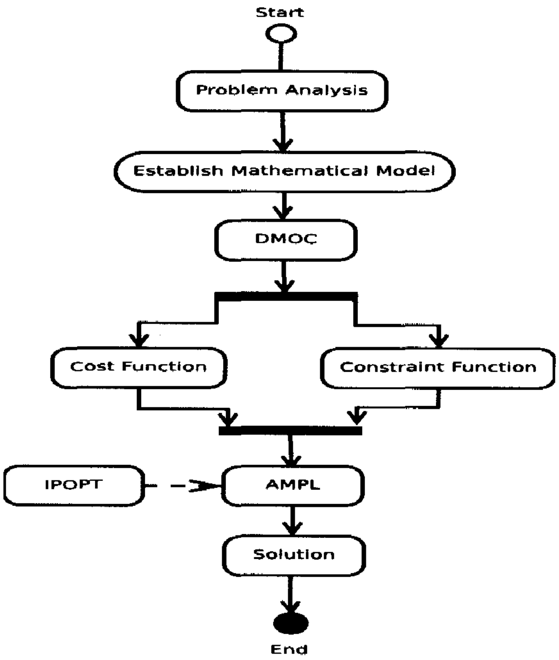


FIGURE 36 – A DMOC procedure to solve an optimization problem.

language AMPL. In particular, the user-defined function is involved with AMPL to solve more complicated problem. The glider trajectory generation example uses DMOC to solve an optimization problem with AMPL and IPOPT.

1. IPOPT

IPOPT as an open source nonlinear programming solver is invented by Dr Andrew Wachter at Carneige Mellon University. It is a primal-dual interior point [74] algorithm with a filter line-search method. IPOPT has been proved attractable using CUTEr test set (954 problems), compared with other two interior-point optimization codes KNITRO and LOQO [75]. In IPOPT, an original optimal control problem is transformed into a sequence of barrier (interior-point) problems for a decreasing sequence of barrier parameters converging to zero. IPOPT includes a line-search filter method with the feasibility restoration phase, second-order corrections which are supposed to improve the proposed step if a trial point

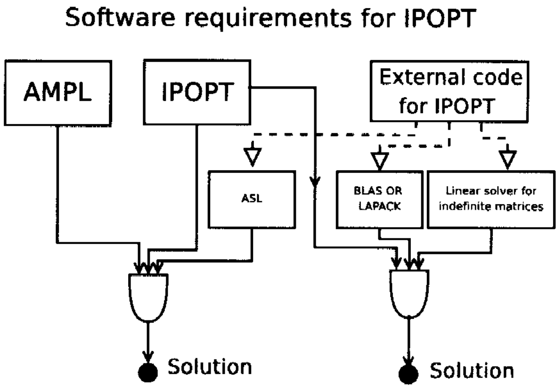

Software requirements for IPOPT
AMPL
IPOPT
External code for IPOPT
ASL
BLAS OR LAPACK
Linear solver for indefinite matrices
Solution
Solution

FIGURE 37 – Software requirements for IPOPT, where ALS is referred to AMPL Solver Library, BLAS represents Basic Linear Algebra Subroutines, LAPACK means Linear Algebra PACKage, one Liner solver for indefinite matrices can be MA27, MA57 or other solvers, the details is described in [7].

has been rejected, and initial correction of the Karush-Kuhn-Tucker matrix which is the necessary optimal condition for nonlinear programing. For the IPOPT algorithm details, see [75].

IPOPT package is available from COIN [76] under the Common Public License. The user can download and use it free of charge even for commercial purposes. Some third party components are required for the execution of IPOPT, these components consist of BLAS (Basic Linear Algebra Subroutines), LAPACK (Linear Algebra PACKage), a sparse symmetric indefinite linear solver such as MA27 or other one. While only ASL (AMPL Solver Library) is required for using with AMPL. The software sources including the dependent solvers are located in the website, the detailed procedure to download and install IPOPT can be found in the IPOPT manual [7].

The user should model problem in a nonlinear programing formulation which can be interfaced with IPOPT through code such as C++, C, or Fortran. For programming problems in C++ interfacing with IPOPT, the user must provide the Jacobian matrix, and Hessian matrix which may be approximated by setting up the IPOPT option "hessian_approximation" as "limited-memory". The eight functions need to be implemented to define the problem and supply the information, the eight functions are get_nlp_info(), get_bounds_info(), get_starting_point(), eval_f(), eval_g(), eval_jac_g(), eval_h(), finalize_solution() separately. As their names indicate, the functions provide the IPOPT with the necessary information like the number of variables, the bounds, the starting points, the constraints Jacobian, the Hessian of the Lagrangian for the solver to generate the solution for the problem. But the difficulty exists in the Jacobian and Hessian parts. The Hessian matrix can be approximated and this eval_h() can be disregarded. Finally it is supposed to provide the IPOPT with the Jacobian of the constraints, which is not easy. For example, in this problem, the constraints have 104 equations, and the number of the variables is 151, therefore, the Jocabian of the constraints has the dimension of 104×151, every element in the matrix needs to be specified, even there is some kind of principle implicit in these elements, it is still not easy to avoid the mistakes for constructing the Jacobian matrix in the programming.

Therefore, the easier way for IPOPT to solve the problem is to interface it with AMPL, because AMPL automatically provide some necessary information to IPOPT to solve the problem, the information include and not limited to the Jacobian, Hessian matrix.

2. AMPL

AMPL, developed in Bell Laboratories, is a comprehensive and powerful algebraic modeling language for linear and nonlinear, continuous or discrete system optimization problems. It is user-friendly, making the user focus on the modelling of the problem, not the technical details for programming. All the variables, parameters, cost functions, constraint functions are defined intuitively and straightforward. The main difference between AMPL with other programming languages such as C or Fortran are the expressions of the variables. In AMPL, "set" and "index" are used to invoke the specific variable. On the other hand, the mathematical expression is generally adapted from an advanced programming language, for example, "sum" or ">" and so on as arithmetic or logical operators are used. Since AMPL is based on algebraic expressions of constraints and objectives. Its syntax is easily learned by referring to the manual [77] or through some examples [78]. With the AMPL scripture of the program, the AMPL translator can read the optimization model and data provided through the language. The seven logic phases are executed like *parse, read data, compile, generate, collect, presolve, out*. The AMPL needs call the solvers to generate the solution for the formulated problem.

3. Implementation Details

To implement, the first step, download IPOPT from COIN [76]. The latest (Apr 26, 2008) C++version of IPOPT tarball is *Ipopt*3.4.0.*tgz*. Assume the tarball is downloaded to the folder *Program/IPOPTtutorial*. Unpack the archive file by *gunzip IPOPT*3.4.0.*tgz*, resulting into *IPOPT*3.4.0.*tar*. Using *tar xvf IPOPT-*3.4.0.*tar*, the tarball is extracted into *IPOPT*3.4.0.*tar*. For convenience, the name of the directory *IPOPT*3.4.0 to *CoinIpopt*. According to Figure IV.B.1, IPOPT needs a few external packages to make it work, including AMPL solver library–ASL, basic linear algebra subroutines–BLAS or Linear Algebra Package–LAPACK, a linear solver for symmetric indefinite matrices such as MA27 or MA57. If IPOPT is used with AMPL as it is in this example, only ASL is required. However, IPOPT can work independently from AMPL, so the procedure to download BLAS, LAPACK, MA27 is listed in the following by utilizing the scripts included in the IPOPT distribution.

- *cd CoinIpopt/ThirdParty/Blas* go to the Blas directory
- *./get.Blas* run the script to download BLAS from the Netlib Repository, after succession, the message "Done downloading the source code for BLAS" appears.
- *cd ../Lapack* go to the Lapack directory
- *./get.Lapack* download Lapack, get the message "Done downloading the

source code for LAPACK".

- *cd ../ASL* go to the ASL directory
- *./get.ASL* download ASL, get the message "Done...".

For the sparse symmetric linear solver MA27, search and get MA27. save *ma27ad.f* to *CoinIpopt/ThirdParty/HSL*. As indicated, other linear solver for symmetric indefinite matices instead of MA27. After the third party codes are installed, IPOPT needs to be compiled and installed by the generally command.

- *cd CoinIpopt*
- *./configure* get the message "configure: Configuration of Ipopt successful,configure: Main configuration of Ipopt successful"
- *make*
- *make install*

After IPOPT is successfully installed in *CoinIpopt/Ipopt*, begin to test the examples to make sure it work. For instance, if go to */examples/Cpp_example*, type *make*, then *./cpp_example*, the screen output should be "Optimal Solution Found.*** The problem solved in six iterations!*** The final value of the objective function is -4.000000e+00". Generally, it means that IPOPT is ready to use for solving your optimization problem. The easiest way to make IPOPT solve an optimization problem is to make it work with AMPL, even also program problems in c, c++ or Fortran language. In this paper, we are considering the problem to make IPOPT work with AMPL. Firstly, AMPL can be downloaded from the web without any charges if the variables are less than 300. In this case, the experimental system is HP Pavilion a1430n, Memory 2.0 GB, AMD Athlon(tm) 64 × 2 Dual Core Processor 3800+. The operating system is Ubuntu 7.10, Kernel Linux 2.6.22-14-386. Thus "Intel (Pentium-compatible) PCs running Linux" AMPL is downloaded to *Ipopt/examples/AMPLex*. Using *gunzip ampl.gz* to uncompress the file into *ampl*, by typing *chmod + x ampl* to make sure to have the privilege to execute the AMPL. Then AMPL is ready to use when modeling the problem in AMPL and the problem is in the format of *test.mod* in which the solver is specified as IPOPT. *./ampl test.mod* is used to solve the problem, the output can be saved into a file and shown on the screen.

The useful feature of AMPL is that it can include user-defined function as an externally added function to solve more complex problems. In order to make the user-defined functions work with AMPL, the "funcadd.c" should be download from the server [78], and modify it according to your purpose, basically the user needs to embeded his or her own program to the downloaded function making the user-defined function work like the example function. Then compile the "funcadd.c" by the different makefile which is dependent on the work station where the program is supposed to execute. Download the makefile from AMPL website and modified it according to the specific system. The Makefile.Linux listed in the appendix in the appendix is tested successfully in Ubuntu 7.10, Kernel Linux 2.6.22-14-386, Memory 2.0 GB, AMD Athlon(tm) 64×2 Dual Core Processor 3800+. After the funcadd.c is compiled by make f Makefile.Linux, the amplfunc.dll will be created, now the user-defined function is ready to be called during the optimization process.

AMPL can hook with different kinds of solvers such as ACRS, MINOS, NPSOL, IPOPT and so onto generate the solution for the optimization problem. The "solve" command in AMPL language make AMPL send the problem information to the solver which is regarded as a separate program, then read the solution back from the solver. The files for the communication between AMPL and the solver is called *stub.suffix* [79]. At the beginning, the initial file from AMPL is *stub.nl* which describe the problem information, after solver received this information and with the specified toleration and iteration parameters, the solver write the solution or resulting information to a file named *stub.sol*. Practically, in order to make AMPL work with IPOPT, the user just need to install AMPL to the right directory, then specify solver option in AMPL program as "option solver IPOPT" at the beginning of the program.

4. An Application Example

The following example is shown how DMOC work with IPOPT and AMPL to solve the optimal control problem. A dynamical glider is simulated moving in Monterey Bay from $(-122.1780, 36.8557)$ to $(-122.2420, 36.6535)$ which represents the position in degrees (Longitude and Latitude), it is controlled by the gyroscopic forces.

The AMPL program listed in the following with the solver specified as IPOPT. For calculation purpose, the position unit is transformed into centimeter based on the reference point as $(-122.3246, 36.5658)$. The ocean current flows are modeled as time-varying 2D B-spline model, the optimal trajectory generation with NTG for the glider in this model has been presented in [8].

In AMPL, the modeled problem is shown as follows:

- Cost function:

$$\text{minimize force\_energy:}$$

$$\text{sum } \{\text{j in 0..N-2}\} 0.5 * (f_1^+[j] * f_1^+[j] +$$

$$f_1^-[j] * f_1^-[j] + f_2^+[j] * f_2^+[j] + f_2^-[j] * f_2^-[j]) * h;$$

- Start and final Constraints:

$$\text{subject to x\_start:} q_1[0] = a_x;$$

$$\text{subject to y\_start:} q_2[0] = a_y;$$

$$\text{subject to x\_destination:} q_1[N-1] = b_x;$$

$$\text{subject to y\_destination:} q_2[N-1] = b_y;$$

- Trajectory Constraints:

$$\text{subject to Euler\_Lagrange\_x } \{\text{j in 0..N-3}\}:$$

$$-KEq1p[j+1] + KEq1p[j] + 0.5 * h * (KEq1[j] + KEq1[j]) +$$

$$0.5 * h * (Vq1[j+1] + Vq1[j]) + f_1^+[j] + f_1^-[j] = 0;$$

subject to Euler_Lagrange_y {j in 0..N-3} :

$$-KEq2p[j+1] + KEq2p[j] + 0.5 * h * (KEq2[j] + KEq2[j]) +$$

$$0.5 * h * (Vq2[j+1] + Vq2[j]) + f_2^+[j] + f_2^-[j] = 0;$$

where f_1^-, f_2^-, f_1^+ and f_2^+ are the left and right discrete forces [9]. for the components of the gyroscopic force.F_{gyr}(15). N is the number of knots in the trajectory. a_x,a_y,b_x,and b_y are the starting and destination point, it has been transformed in the program to the centimeter assuming the radius of the earth is $6378km$ and the earth is a perfect sphere. The trajectory constraints are introduced because the glider is controlled by the gyroscopic force and its motion is satisfied with Lagrange-d'Alembert principle. These constraints are called discrete Euler-Lagrangian equations [9]. In the trajectory constraints, $KEq1p$, $KEq2p$ is the derivative of the glider kinetic energy according to $\dot{q}_1$, $\dot{q}_2$ respectively. $Vq1$, $Vq2$ is the potential energy of the glider. The index of all these variables shows the states are discrete in the program. For the glider travels in the ocean current. The function is defined in AMPL in the way shown in the following:

- Define function:

  function splineinfo;

- Call function to retrieve the ocean current velocities:

  var u $\{i\ in\ VEL\_NODES\}$

  $= \text{splineinfo}(x[i], y[i], h * i/3600, u, v);$

  var v $\{i\ in\ VEL\_NODES\} = v;$

- Discrete forces are connected with currents:

  var $f_1^+\{i\ in\ VEL\_NODES\}$

  $= 0.5 * h * (-taum[i] * (q2p[i] - v[i]));$

  var $f_1^-\{i\ in\ VEL\_NODES\}$

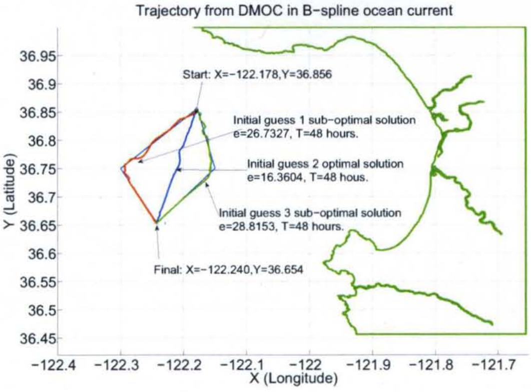


FIGURE 38 – Dynamical glider trajectories in the 3D B-spline current model.

TABLE 6
DMOC SOLUTIONS FOR A DYNAMICAL GLIDER IN THE 3D B-SPLINE CURRENT MODEL.

$DMOC$	$Interval$	$T(hours)$	$Iter$	$Time(s)$	$Energy\,Cost$
$Guess$ 1	40	42.51	3000	23.10	7.9615
$Guess$ 2	42	30.05	3000	43.30	62.2179
$Guess$ 3	40	43.22	3000	23.10	16.192

$$= 0.5 * h * (-taum[i] * (q2p[i] - v[i]));$$

$$\text{var } f_2^+\{i \; in \; VEL\_NODES\}$$

$$= 0.5 * h * (taum[i] * (q1p[i] - u[i]));$$

$$\text{var } f_2^-\{i \; in \; VEL\_NODES\}$$

$$= 0.5 * h * (taum[i] * (q1p[i] - u[i]));$$

where $VEL\_NODES = \{0, ...N-1\}$, h is the step size of optimization, $taum$ is the control force, $q2p$, $q1p$ is the derivative approximation of $q1$ and $q2$. The index of variables show that the states are discrete.

By combining DMOC, IPOPT, AMPL and time-varying 2D B-spline ocean current model, the glider trajectory is generated and shown in Figure 38. As shown in TABLE 6, DMOC has successfully generated the local solutions for the optimal control of the glider with a complex ocean current model. It illustrates the promising aspects of DMOC methodologies combined with IPOPT to solve other optimization control problems. Furthermore, an efficient method is needed to choose a better solution from local solutions. The obtained solution will be an approximate global solution for the optimal control problem.

C. Summary

In this chapter, a tutorial on solving the optimal control problems with DMOC is presented. It is shown that DMOC combined with AMPL and IPOPT can solve complex optimal control problems. Especially, it is shown that user-defined functions can be invoked in this procedure. The fundamentals of IPOPT and AMPL are explained and procedure to solve problems is presented. As an example, a dynamic glider is simulated moving in Monterey Bay California where the ocean current is modeled as time varying 2D B-spline function. The minimizing energy local solution trajectories are obtained by DMOC methodology. Consequently, this tutorial proposes a feasible approach and procedure to solve optimal control problems with the available resources including DMOC methodology, the open source IPOPT, the AMPL with a free student version with 300 variables limit.

CHAPTER V

COMPARISON OF DMOC AND NTG

In this chapter, two state-of-the-art optimal trajectory generation methodologies as DMOC and NTG are analyzed and compared. DMOC is a recently developed methodology to solve optimal control problems for mechanical systems. It is based on a direct discretization of the Lagrange-d'Alembert principle while NTG is based on a combination of differential flatness, spline theory and sequential quadratic programming. Theoretical foundations and results for comparisons are presented with application to a dynamic glider. In a simple ocean current example, DMOC with discrete Euler-Lagrange constraints generates local optimal solutions with different initial guesses while NTG is also generating similar solutions with more computation time and less energy consumption. Furthermore in a more complex ocean current model, optimal solutions from DMOC also cost less energy and computation time than the ones from NTG. In both cases, DMOC optimal solutions are shown to cost less energy and less computation time than NTG optimal solutions. The cost functions are the integral of control forces over time, nonlinear constraints are direct discrete Euler-Lagrange equations for DMOC, continuous ones for NTG.

A. Discrete Mechanics and Optimal Control

Discrete Mechanics and Optimal Control (DMOC) [9] is based on a direct discretization of the Lagrange-d'Alembert principle. For comparison, NTG uses the continuous version of Euler-Lagrange equations as constraints.

1. Discrete Cost Function

The cost function (50) over a time slice $[kh, (k+1)h]$ is approximated as

$$C_d(q_k, q_{k+1}, f_k, f_{k+1}) \approx \int_{kh}^{(k+1)h} C(q, \dot{q}, f)dt \tag{63}$$

The integral in the cost function can be approximated by some standard methods such as the Midpoint Rule [9]. Thus the overall cost function becomes as:

$$J_d(q_d, f_d) = \sum_{k=0}^{N-1} C_d(q_k, q_{k+1}, f_k, f_{k+1}) \tag{64}$$

2. Discrete Lagrange-d'Alembert Principle

The innovative part of DMOC is to exploit the variational structure directly. Instead of first deriving the Euler-Lagrange equations (equations of motion) for the system to get the optimal solution, it uses a global discretization of the states and the controls by the discrete Lagrange-d'Alembert principle to obtain equality constraints, then the problem is transformed into a finite dimensional nonlinear optimization problem.

For a trajectory generation problem shown with the cost function as (50), constraint functions as (52) and (51), the time for system states is discretized as $0, h, 2h, ...Nh = t_f$, where h is the step size and $N \in \mathbb{N}$. The continuous state $q(t)$ and the continuous force $f(t)$ is approximated by discrete states $q_d(kh)$ and forces $f_d(kh)$.

Through direct discretization, Lagrangian in Euler-Lagrange equation (52) is approximated over a time slice $[kh, (k+1)h]$ by a discrete Lagrangian Ld.

$$L_d(q_k, q_{k+1}) \approx \int_{kh}^{(k+1)h} L(q(t), \dot{q}(t))dt, \tag{65}$$

At the same time, the virtual work in (52) also can be approximated as

$$f_k^- \cdot \delta q_k + f_k^+ \cdot \delta q_{k+1} \approx \int_{kh}^{(k+1)h} f(t) \cdot \delta q(t)dt, \tag{66}$$

where f_k^-, f_k^+ are called left and right discrete force respectively.

$$f_k^- = f(kh) \cdot h/2; \tag{67}$$

$$f_k^+ = f((k+1)h) \cdot h/2 \tag{68}$$

The discrete version of Lagrange-d'Alembert principle (52) requires that

$$\delta \sum_{k=0}^{N-1} L_d(q_k, q_{k+1}) + \sum_{k=0}^{N-1} (f_k^- \cdot \delta q_k + f_k^+ \cdot \delta q_{k+1}) = 0. \tag{69}$$

As it is shown in IV, the discrete Lagrange-d'Alembert principle is derived as

$$D_1 L_d(q_k, q_{k+1}) + D_2 L_d(q_{k-1}, q_k) + f_{k-1}^+ + f_k^- = 0, \tag{70}$$

where $k = 1 \ldots N-1$, called the forced discrete Euler-Lagrange equations. Other constraints such as (51) are also discretized with corresponding discrete states and forces, see [9] for details. Discrete boundary conditions at $t_0 = 0$ and $t_f = Nh$ are expressed as

$$\begin{aligned} D_2 L(q_0, \dot{q}_0) + D_1 L_d(q_0, q_1) + f_0^- &= 0, \\ -D_2 L(q_N, \dot{q}_N) + D_2 L_d(q_{N-1}, q_N) + f_{N-1}^+ &= 0. \end{aligned} \tag{71}$$

B. Nonlinear Trajectory Generation

1. Problem Formulation

NTG methodology is based on a combination of nonlinear control theory, spline theory, and Sequential Quadratic Programming. NTG transforms optimal control problem into Nonlinear Programming Problem (NLP). It is then solved by NPSOL [80] [44]. In NTG the collocation method [72] is to choose a finite-dimensional space of candidate solutions and a number of points (collocation points) in the domain, and to select a solution which satisfies given cost and constraints equations at the collocation points. In order to compare DMOC with NTG, the cost and constraint functions should be the same. The optimal control problem for NTG is to obtain $f(t)$ to minimize a cost function.

$$J(q,f) = \int_{t_0}^{t_f} C(q(t),\dot{q}(t),f(t))dt \tag{72}$$

At the same time, the motion of $q(t)$ of the mechanical system from $(q^{t_0},\dot{q}^{t_f})$ to a state $(q^{t_0},\dot{q}^{t_f})$ is to satisfy the Lagrange-d'Alembert principle [9]. This constraint (52) can be expressed as

$$\int_{t_0}^{t_f} (\frac{\partial L}{\partial q}\delta q + \frac{\partial L}{\partial \dot{q}}\delta \dot{q})dt + \int_{t_0}^{t_f} f(t)\delta q(t)dt = 0 \tag{73}$$

As shown in III.B.1, the continuous Euler-Lagrange equation is obtained in the following.

$$\frac{\partial L}{\partial q} - \frac{d}{dt}\frac{\partial L}{\partial \dot{q}} + f(t) = 0 \tag{74}$$

Other constraints are listed as (6):

$$\begin{array}{lllll} \text{Initial} & lb_0 & \leq & \Psi_0(q(t_0),\mathbf{f}(t_0),t_0) & \leq & ub_0 \\ \text{Trajectory} & lb_t & \leq & \Psi_t(q(t),\mathbf{f}(t),t) & \leq & ub_t \\ \text{Final} & lb_f & \leq & \Psi_f(q(t_f),\mathbf{f}(t_f),t_f) & \leq & ub_f \end{array} \tag{75}$$

where $q(t)$ is the state of the system and $f(t)$ is the control input. The constraints compose of initial constraints $\Phi_0(.)$, trajectory constraints, $\Phi_t(.)$, and final constraints, $\Phi_f(.)$. lb and ub are lower and upper bounds for the constraint functions and t_0, t_f are the initial and final time. If the cost function and constraints are evaluated at discrete points in the interval $[t_0, t_f]$, it is possible to transform the optimization problem, defined by (72) and (74), (6), into the following NLP problem in C_j:

$$\min_{\vec{C}\in\mathbb{R}^p} F(\vec{C})$$

subject to

$$LB \leq G(\vec{C}) \leq UB$$

where $\vec{C} = [C_1 \ \cdots \ C_p]^T$, $F(\vec{C})$ is the transformed cost function, and $G(\vec{C})$ are the transformations of the constraints, with LB and UB as the lower and upper bounds, respectively. The discrete points, C_i, at which cost and constraints are evaluated, are called *collocation points*.

2. Procedure in NTG

Three steps are required in the NTG algorithm. The first step is to exploit any differential flatness of the system to find a new set of outputs of the system so that the system dynamics can be mapped down to a lower-dimensional space, with the property that all the states and controls of the original system can be recovered from the new lower-dimensional representation. The differentially flat system [81] means the system states and inputs can be determined from a new set of outputs without integration. Suppose that the system has states $q1 \in R^n$, and inputs $f \in R^m$ then the system is flat if new outputs can be found such that $q_2 \in R^m$ of the form $q_2 = q_2(q_1, f, \dot{f}, ..., f^{(p)})$ such that $q_1 = q_1(q_2, \dot{q_2}, ..., q_2^{(k)})$, and $f = f(q_2, \dot{q_2}, ..., q_2^{(k)})$, where p, k are constant variables. The second step is to further represent these outputs in terms of the B-spline functions:

$$z_i(t) = \sum_{j=1}^{p_i} B_{j,r_i}(t) C_j^i$$

where p_i is the number of free parameters C_j^i (coefficients of the B-spline functions). B_{j,r_i} are B-spline basis functions. The basis functions [34] are defined as:

$$B_{j,0} = \begin{cases} 1, & \text{if } t_j \le t < t_{j+1} \\ 0, & otherwise \end{cases} \tag{76}$$

$$B_{j,n}(t) = \frac{t - t_j}{t_{j+n} - t_j} B_{j,n-1}(t) + \frac{t_{j+n+1} - t}{t_{j+n+1} - t_{j+1}} B_{j+1,n-1}(t) \tag{77}$$

Lastly, to solve the coefficients of the B-spline functions, C_j^i, with the sequential quadratic programming solver NPSOL.

C. DMOC versus NTG

Two examples are presented to compare DMOC and NTG. One example is a dynamical glider in a simplified current situation, and the other example shows trajectories of this glider in a complex B-spline ocean current model. In both cases the glider is controlled by gyroscopic forces. DMOC and NTG formation have the same cost function, the same constraints, and the same nonlinear programming solver.

1. A Glider in the Simple Current Model

Considering the example presented in [82], a dynamical glider shown in Figure 4 modelled as (78) is moving from the point $q_1 = 10, q_2 = 0$ to $q_1 = 15, q_2 = 2$ with the unit of centimeter in ocean current where q_1, and q_2 represent x and y directions in a Cartesian coordinate system. The initial relative velocities are $\dot{q}_1 = -10, \dot{q}_2 = -10$ with the unit as centimeter/second, and it is controlled by gyroscopic force F_{gyr} (79).

$$\begin{aligned} \ddot{q}_1 &= -\tfrac{d\theta}{dt}\left(\dot{q}_2 - v\right) + \dot{u} \\ \ddot{q}_2 &= \tfrac{d\theta}{dt}\left(\dot{q}_1 - u\right) + \dot{v} \end{aligned} \tag{78}$$

$$F_{gyr} = \begin{pmatrix} f_1 = -\tau\left(\dot{q}_2 - v\right) \\ f_2 = \tau\left(\dot{q}_1 - u\right) \end{pmatrix} \tag{79}$$

where θ is the orientation of the glider shown in Figure 4, u and v are components of the ocean current velocity in the q_1 and q_2 direction, respectively. $\ddot{q}_1$, $\ddot{q}_2$, $\dot{q}_1$, $\dot{q}_2$ are accelerations and velocity of the glider in x and y direction, respectively, and τ is a control input.

a. Problem Formulation in DMOC The cost function of this problem is given as:

$$J = \int_{t_0}^{t_f} \parallel F_{gyr} \parallel^2 dt \tag{80}$$

Since the control input is a gyroscopic force, the cost function is an integral of these forces over the operation time $[t_0, t_f]$. With (15), it can be expressed as

$$J = \int_{t_0}^{t_f} \left(f_1^2 + f_2^2\right) dt \tag{81}$$

Discrete form of the cost function can be derived as:

$$J_d = \sum_{k=0}^{N-1} (f_1(k)^2 + f_2(k)^2)h \tag{82}$$

where N is number of intervals in the trajectory and h is the step size in the trajectory. In the DMOC problem formulation, velocity at knot j of the trajectory in q_1, q_2 are correspondingly approximated as

$$\begin{aligned} q_1p[j] &= (q_1[j+1] - q_1[j])/h \\ q_2p[j] &= (q_2[j+1] - q_2[j])/h \end{aligned} \tag{83}$$

$$\begin{aligned} f_1(j) &= -\tau[j]\,(q_2p[j] - v[j]) \\ f_2(j) &= \tau[j]\,(q_1p[j] - u[j]) \end{aligned} \tag{84}$$

$u[j]$, and $v[j]$ are the current velocities in the q_1 and q_2 directions, respectively.

Euler-Lagrange equations as the constraints are listed in (61) and (62). Specifically, the Lagrangian L of the glider is the difference between the kinetic energy KE and the potential energy PE.

$$L(q(t), \dot{q}(t)) = KE(q(t), \dot{q}(t)) - PE(q(t), \dot{q}(t)) \tag{85}$$

$$\begin{aligned} KE_{q_1}(q_1(t), \dot{q}_1(t)) &= \frac{1}{2}m(\dot{q}_1(t) - u)^2 \\ KE_{q_2}(q_2(t), \dot{q}_2(t)) &= \frac{1}{2}m(\dot{q}_2(t) - v)^2 \end{aligned} \tag{86}$$

$$\begin{aligned} PE_{q_1}(q_1(t), \dot{q}_1(t)) &= 0 \\ PE_{q_2}(q_2(t), \dot{q}_2(t)) &= 0 \end{aligned} \tag{87}$$

According to the discrete Euler-Lagrange equations (61), trajectory constraints are

$$\begin{aligned} -m(q_1p[j+1] - u[j+1]) + m(q_1p[j] - u[j]) + f_1^+[j] + f_1^-[j+1] &= 0 \\ -m(q_2p[j+1] - v[j+1]) + m(q_2p[j] - v[j]) + f_2^+[j] + f_2^-[j+1] &= 0 \end{aligned} \tag{88}$$

where the left and right discrete forces are

$$\begin{aligned} f_1^-[j] = f_1^+[j] &= -\tau[j](q_2p[j] - v[j])h/2 \\ f_2^-[j] = f_2^+[j] &= \tau[j](q_1p[j] - u[j])h/2 \end{aligned} \tag{89}$$

They are different from $f_1[j]$, $f_2[j]$. The initial boundary constraints are listed as:

$$\begin{aligned} q_1[0] - q_{1ini} &= 0 \\ q_2[0] - q_{2ini} &= 0 \\ -m(q_1p[0] - u[0]) + f_1^-[0] + mq_{1inip} &= 0 \\ -m(q_2p[0] - v[0]) + f_2^-[0] + mq_{2inip} &= 0 \end{aligned} \tag{90}$$

The final conditions are listed as:

$$\begin{aligned} q_1[N-1] - q_{1final} &= 0 \\ q_2[N-1] - q_{2final} &= 0 \\ -m(q_1p[N-1] - u[N-1]) + f_1^+[N-1] + mq_{1finp} &= 0 \\ -m(q_2p[N-1] - v[N-1]) + f_2^+[N-1] + mq_{2finp} &= 0 \end{aligned} \tag{91}$$

where (q_{1ini}, q_{2ini}) and (q_{1final}, q_{2final}) are initial and final positions of the glider at q_1 and q_2, which are $(10, 0)$ and $(15, 2)$. (q_{1inip}, q_{2inip}), (q_{1finp}, q_{2finp}) are initial and final velocities of the glider. m is the mass of the glider. (q_{1inip}, q_{2inip}) are $(-10, -10)$ correspondingly. $u(0)$, and $v(0)$ are ocean current velocities at the initial time. In addition to the discrete Euler-Lagrange equation constraints, the variables are also constrained as

$$\begin{aligned} -100 \le q_1[j] \le 100; \\ -100 \le q_2[j] \le 100; \\ -100 \le \tau[j] \le 100; \\ 1 - \epsilon \le t_f - t_0 \le 1 + \epsilon; \end{aligned}$$

ϵ is a small number. For this example, the velocity of the current in x-direction u is equal to the value of x multiplied by 0.1, the velocity v in y-direction is assumed to be 0. The glider is controlled by gyroscopic forces given in (84), and an animation to show that the glider is generating an optimal trajectory by DMOC.

To generate optimal trajectories with DMOC, the problem modelled in AMPL is solved by a nonlinear programming solver NPSOL which is the same as the NTG

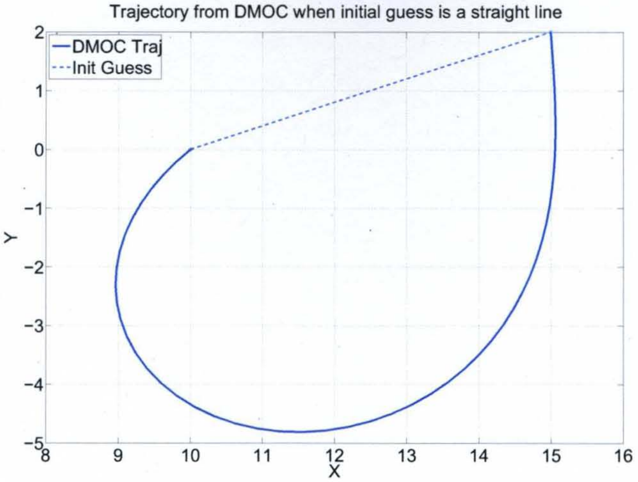


FIGURE 39 – The DMOC solution when the initial guess is a straight line.

solver. The default NPSOL options [44] are used in both DMOC and NTG solutions of the problem. The number of intervals as 50 is the same as the one defined in NTG. When the initial guess is a straight line connecting the start and the destination points, an optimal solution from DMOC is shown in Figure 39. After 96 iterations, the optimal cost is 4672.54, and the final time is 1 second. The system for the experiments is Ubuntu 7.10, Kernel Linux 2.6.22-14-386, Memory 2.0 GB, AMD Athlon(tm) 64×2 Dual Core Processor 3800+. Figure 40 shows the control input τ in DMOC changes smoothly with the time. In the DMOC trajectory when the initial guess is a straight line, the changes of the coordinates and velocities are smooth and listed in Figure 41.

b. Problem Definition in NTG In this part, optimal trajectory generation problem of the glider is formulated in NTG as listed in the following. The cost function J as in (80):

$$J = \int_{t_0}^{t_f} \left(f_1^2 + f_2^2\right) dt \tag{92}$$

with (15), the cost function becomes

$$J \ = \int_{t_0}^{t_f} (-\tau(\dot{q}_2 - v))^2 + (\tau(\dot{q}_1 - u))^2 dt.$$

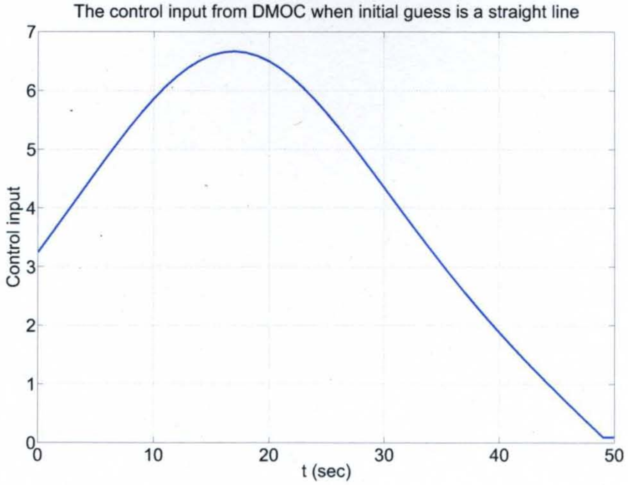
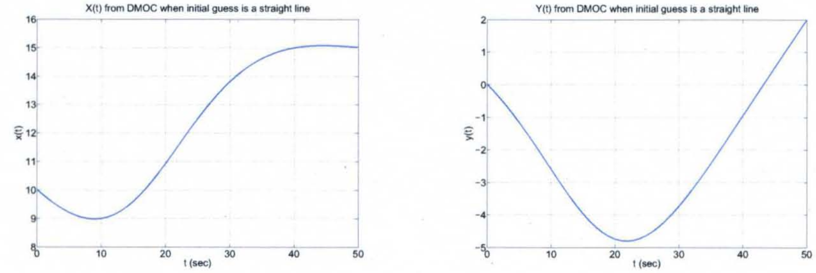
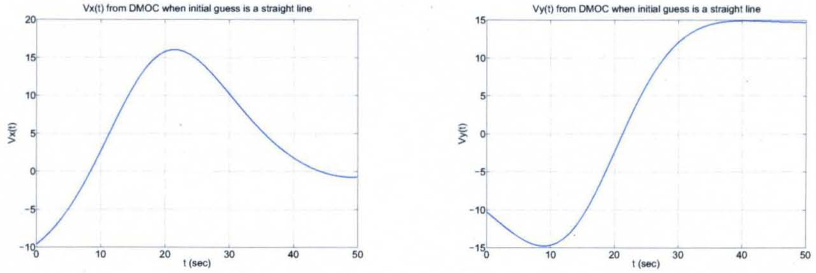


FIGURE 40 – The control input τ in DMOC when the initial guess is a straight line.



a) x(t)

b) y(t)



c) Vx

d) Vy

FIGURE 41 – The DMOC trajectory properties when the initial guess is a straight line.

where t_0, t_f are the start and final time for the trajectory, respectively. Constraints in NTG formulation are given as:

- Trajectory constraints: The glider is controlled by F_{gyr}, and its motion is constrainted by Euler-Lagrange equation:

$$\frac{\partial L}{\partial q} - \frac{d}{dt}\frac{\partial L}{\partial \dot{q}} + f(t) = 0 \tag{93}$$

  where q can be q_1 or q_2, $f(t)$ should be f_1 and f_2, correspondingly. The deduced continuous Euler-Lagrange equations are represented in q_1 and q_2 directions as

$$\frac{\partial L_1}{\partial q_1} - \frac{d}{dt}\frac{\partial L_1}{\partial \dot{q}_1} + f_1(t) = 0; \tag{94}$$

$$\frac{\partial L_2}{\partial q_2} - \frac{d}{dt}\frac{\partial L_2}{\partial \dot{q}_2} + f_2(t) = 0 \tag{95}$$

  Other constraints are listed in the following.

- Initial Constraints:

$$10 - \epsilon \leq q_1(t_0) \leq 10 + \epsilon$$

$$0 - \epsilon \leq q_2(t_0) \leq 0 + \epsilon$$

$$-10 - \epsilon \leq \dot{q}_1(t_0) \leq -10 + \epsilon$$

$$-10 - \epsilon \leq \dot{q}_2(t_0) \leq -10 + \epsilon$$

- Final Constraints:

$$15 - \epsilon \leq q_1(t_f) \leq 15 + \epsilon$$

$$2 - \epsilon \leq q_2(t_f) \leq 2 + \epsilon$$

- Other Trajectory Constraints:

$$-100 - \epsilon \leq q_1(t) \leq 100 + \epsilon$$

$$-100 - \epsilon \leq q_2(t) \leq 100 + \epsilon$$

$$-100 - \epsilon \leq \tau \leq 100 + \epsilon$$

$$1 - \epsilon \leq t_f - t_0 \leq 1.0 + \epsilon$$

where t_0, t_f are the start and final time for the trajectory, respectively. Constraints in NTG formulation are given as:

- Trajectory constraints: The glider is controlled by F_{gyr}, and its motion is constrainted by Euler-Lagrange equation:

  $$\frac{\partial L}{\partial q} - \frac{d}{dt}\frac{\partial L}{\partial \dot{q}} + f(t) = 0 \tag{93}$$

  where q can be q_1 or q_2, $f(t)$ should be f_1 and f_2, correspondingly. The deduced continuous Euler-Lagrange equations are represented in q_1 and q_2 directions as

  $$\frac{\partial L_1}{\partial q_1} - \frac{d}{dt}\frac{\partial L_1}{\partial \dot{q}_1} + f_1(t) = 0; \tag{94}$$

  $$\frac{\partial L_2}{\partial q_2} - \frac{d}{dt}\frac{\partial L_2}{\partial \dot{q}_2} + f_2(t) = 0 \tag{95}$$

  Other constraints are listed in the following.

- Initial Constraints:

  $$10 - \epsilon \leq q_1(t_0) \leq 10 + \epsilon$$

  $$0 - \epsilon \leq q_2(t_0) \leq 0 + \epsilon$$

  $$-10 - \epsilon \leq \dot{q}_1(t_0) \leq -10 + \epsilon$$

  $$-10 - \epsilon \leq \dot{q}_2(t_0) \leq -10 + \epsilon$$

- Final Constraints:

  $$15 - \epsilon \leq q_1(t_f) \leq 15 + \epsilon$$

  $$2 - \epsilon \leq q_2(t_f) \leq 2 + \epsilon$$

- Other Trajectory Constraints:

  $$-100 - \epsilon \leq q_1(t) \leq 100 + \epsilon$$

  $$-100 - \epsilon \leq q_2(t) \leq 100 + \epsilon$$

  $$-100 - \epsilon \leq \tau \leq 100 + \epsilon$$

  $$1 - \epsilon \leq t_f - t_0 \leq 1.0 + \epsilon$$

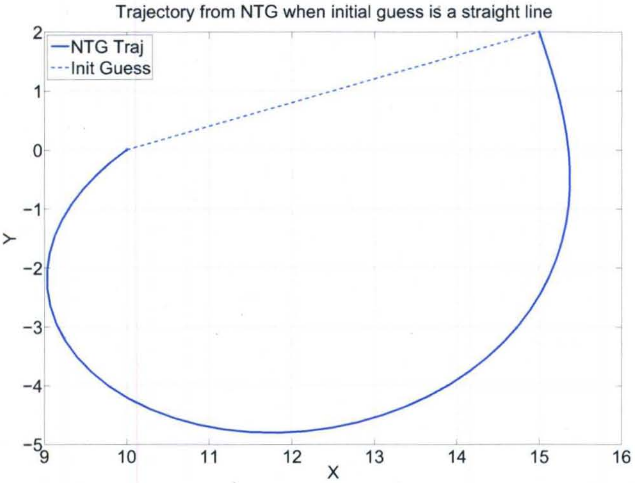


FIGURE 42 – The minimizing-energy trajectory from NTG when the initial guess is a straight line.

where $q_1(t_0)$, $q_2(t_0)$, $q_1(t_f)$, $q_2(t_f)$ are initial and final locations of the glider. $\dot{q}_1(t_0)$ and $\dot{q}_2(t_0)$ are initial velocities of the glider. f_1 and f_2 are two components of the F_{gyr} given in (15).

When NTG, using NPSOL with $linesearch = 1e - 10$ and other default options, is applied to solve this optimal trajectory generation problem with a straight line initial guess, the optimal trajectory shown in Figure 42 is slightly different from the one shown in Figure 39 obtained with DMOC. The final nonlinear objective value is 4163.94 and the final time is 1 sec, the computation time is 43.47 sec. In NTG, the variables q_1, q_2, and τ are specified as 50 intervals, the B-spline order for these three variables are 4, and the smoothness are all two.

c. *Comparisons* When the initial guess is a straight line, the trajectories from NTG and DMOC are similar. DMOC successfully generates an optimal solution while NTG is only generating a sub-optimal solution in which sub-optimal means the solution cannot be improved upon from NTG. DMOC costs less energy and computation time to generate an optimal solution than NTG.

Furthermore, when the initial guesses change, trajectories from DMOC and NTG with initial guesses are shown in Figure 46 through Figure 48.

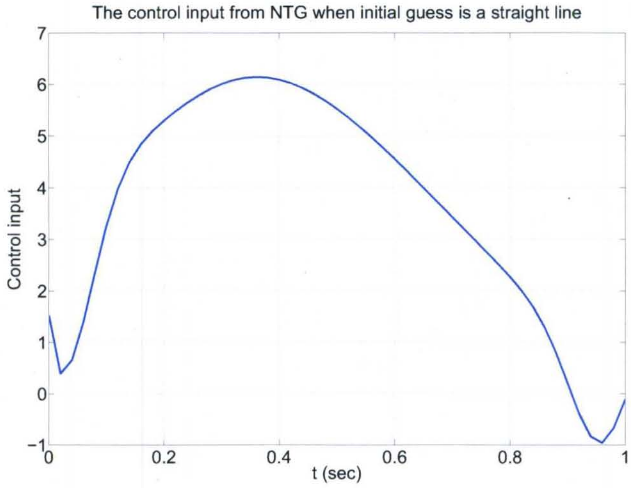
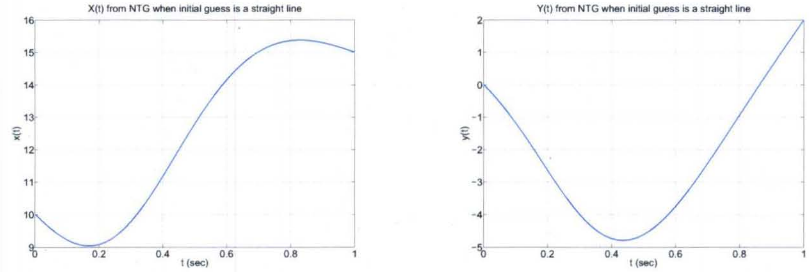
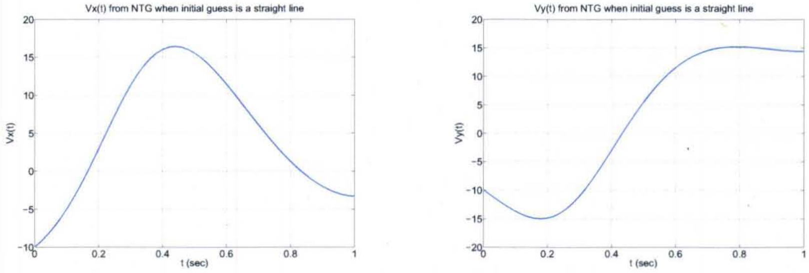


FIGURE 43 – The control input τ in NTG when the initial guess is a straight line.



a) x(t)

b) y(t)



c) Vx

d) Vy

FIGURE 44 – The NTG optimal trajectory properties when the initial guess is a straight line.

TABLE 7
DMOC VS. NTG FOR A GLIDER IN A SIMPLE OCEAN CURRENT MODEL

$DMOC/NTG$	$Interval$	T	$Iter$	$Time(s)$	$Energy\ Cost$
$Guess\ 1$	50/50	1/1	94/144	2.61/43.47	4672.81*/4163.94
$Guess\ 2$	50/50	1/1	43/62	1.65/17.42	5569.49*/4870.12

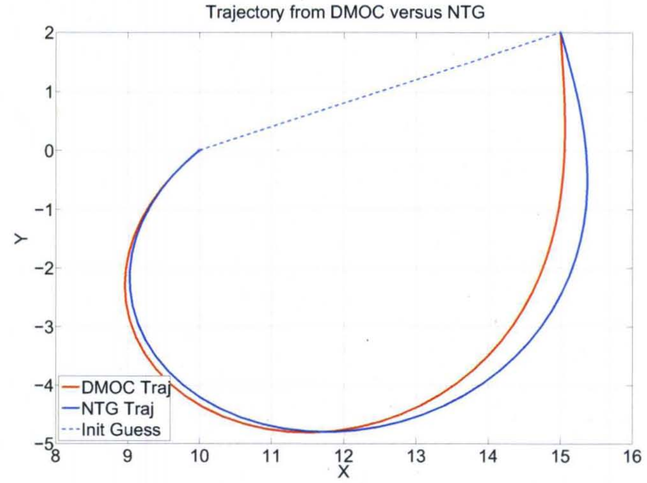


FIGURE 45 – The trajectory from DMOC versus the one from NTG when the initial guess is a straight line.

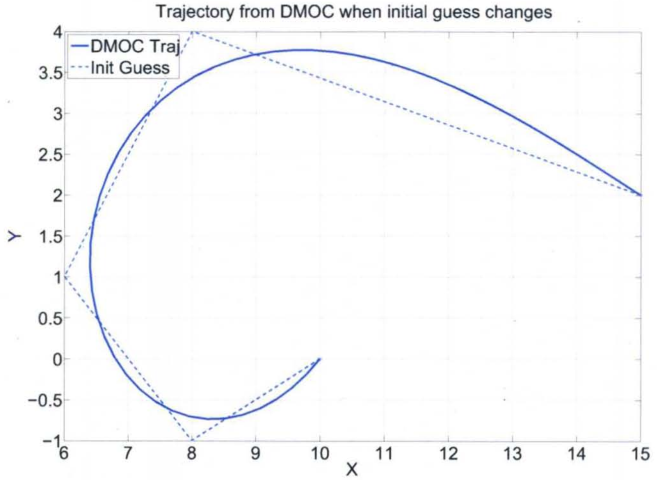


FIGURE 46 – The trajectory from DMOC when initial guess changes.

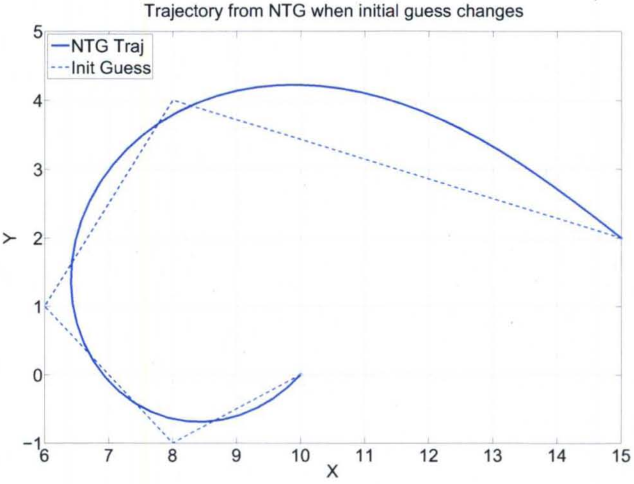


FIGURE 47 – The trajectory from NTG when initial guess changes.

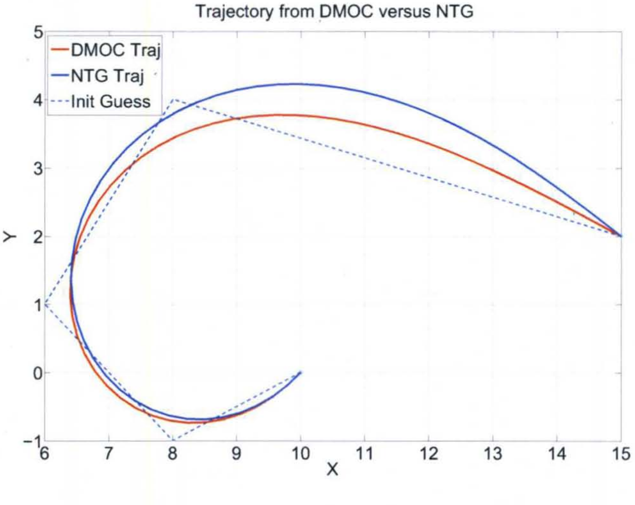


FIGURE 48 – Trajectory from DMOC versus the one from NTG when initial guess changes.

TABLE 7 compares the trajectories found by DMOC versus NTG in terms of interval, final time, iteration number, computation time, and energy cost. In this table, $*$ at the right corner of the data indicates that solution is optimal, otherwise it is sub-optimal. $T = t_f - t_0$ means the final time for the glider to get the destination. $Iter$ means iterations, $Time$ means the time for the program to solve the problem, the unit is seconds. $Guess$ 1 means the initial guess for both cases are the straight line. $Guess$ 2 represents that the initial guess is the curve shown in Figure 48 as the dotted line.

NTG obtained the solution with the defined B-spline variables of q_1, q_2, τ, and T. Their degree of smoothness is $(2,2,2,1)$, B-spline order is $(4,4,4,1)$ in the number of interval is $(50,50,50,1)$. If the interval number, order, and smoothness. If these values are not correctly chosen, NTG cannot generate a satisfactory solution. On the other hand, there is only one adjustable DMOC parameter. That is the number of interval chosen as 50 which is the same as NTG for q_1, q_2, τ. It is interesting to point out that the two trajectories from two initial guesses both in DMOC and NTG are symmetric to each other which are due to the current model.

Even the optimal DMOC solutions from $Guess$ 1 and $Guess$ 2 are used as initial guesses for NTG, NTG solutions do not change, while the iteration numbers are reasonably reduced from 144 (43.47 seconds) to 55 (15.66 seconds) and from 62 (17.42 seconds) to 49 (13.56 seconds) correspondingly. On the other hand, DMOC solutions also do not change when initial guesses are changed to the corresponding NTG solutions from $Guess$ 1 and $Guess$ 2. In this case, the iteration numbers also reduced from 94 (2.61 seconds) to 20 (0.728 seconds)and from 43 (1.65 seconds) to 25 (0.992 seconds) in these two cases.

Then reduce the step size of DMOC optimization by increasing the interval number, the energy costs are correspondingly increasing a little bit, they are listed in TABLE 8. And it seems that the cost is to converge to a definite number. DMOC trajectories seem to converge to a trajectory which is closer to the NTG trajectory. The violations of the NTG constraints are decreasing with the increased intervals.

TABLE 8
DMOC SOLUTIONS WITH DIFFERENT INTERVALS WITH THE SAME INITIAL GUESS (GUESS 3)

$Interval$	$EnergyCost$	$time$	$Iter$	$\dot{q}_1(0)$	$\dot{q}_2(0)$
50	4672.81	4.56	169	−10.0097	−9.9997
60	4674.78	7.44	158	−10.0081	−9.9998
70	4675.96	14.45	183	−10.0070	−9.9998
80	4676.73	23.15	174	−10.0061	−9.9999
90	4677.26	35.63	175	−10.0055	−9.9999
100	4677.64	36.35	135	−10.0049	−9.9999

In TABLE 8, $\dot{q}_1(0), \dot{q}_2(0)$ are the initial velocities obtained from the optimal trajectories. They are initially enforced as $(-10, -10)$. $Guess$ 3 is the initial guess shown in Figure 49, not the ones in Figure 42 and Figure 48.

For different intervals, 6 DMOC trajectories in Figure 49 are shown almost at the same location compared with NTG trajectory. However by checking the DMOC trajectories in more details , they are separated and becoming closer to the NTG trajectory (violations of NTG constraints are decreasing) when the interval number is increasing. On the other hand, shown in TABLE 9 and TABLE 10, NTG solutions with break point number as 100 are satisfied with their own constraints with EuX value from $-8.5980e-6$ to $6.9911e-6$ with the average value as $-1.4239e-7$, and EuY values vary from $-6.2227e-6$ to $-1.2265e-7$ with the average value as $-1.2265e-7$. For other break point numbers, NTG solutions are not successfully generated. EuX and EuY indicate the Euler-Lagrange equation values in x and y directions.

While put the NTG solution (initial guess is a straight line) into the DMOC constraints, the Euler-Lagrange equation in x-direction value is from -0.7640 to 0.5225, the average value is -0.0529. It is from -0.0510 to 0.6925, the average value is 0.0745 in y direction. On the other hand, when the DMOC solution is

TABLE 9
DMOC SOLUTIONS IN NTG CONSTRAINTS VERSUS DMOC CONSTRAINTS (EUX)

$Interval$	$EuX(NTG)/DMOC$		
	min	max	avg
50	$-1.0304/-0.0751$	$3.4748/0.0010$	$0.6393/-0.0017$
60	$-3.7772/-0.0172$	$4.4823/0.0007$	$-0.0087/-0.0004$
70	$-3.2493/-0.0181$	$4.1310/0.0006$	$-0.0320/-0.0003$
80	$-2.8554/-0.0086$	$3.6110/0.0004$	$-0.0385/-0.0001$
90	$-2.5499/-0.0091$	$3.1909/0.0004$	$-0.0509/-0.0001$
100	$-2.3059/-0.0093$	$2.8616/0.0003$	$-0.0604/-0.0001$

TABLE 10
DMOC SOLUTIONS IN NTG CONSTRAINTS VERSUS DMOC CONSTRAINTS (EUY)

$Interval$	$EuY(NTG)/DMOC$		
	min	max	avg
50	$-2.6879/-0.0247$	$1.2539/0.0020$	$-0.3967/0.0001$
60	$-5.5423/-0.0051$	$2.3577/0.0014$	$-0.6828/0.0003$
70	$-4.7531/-0.0053$	$2.0230/0.0011$	$-0.5856/0.0002$
80	$-4.1586/-0.0053$	$1.7760/0.0008$	$-0.5127/0.0002$
90	$-3.7021/-0.0055$	$1.5738/0.0006$	$-0.4561/0.0001$
100	$-3.3354/-0.0055$	$1.4204/0.0005$	$-0.4109/0.0001$

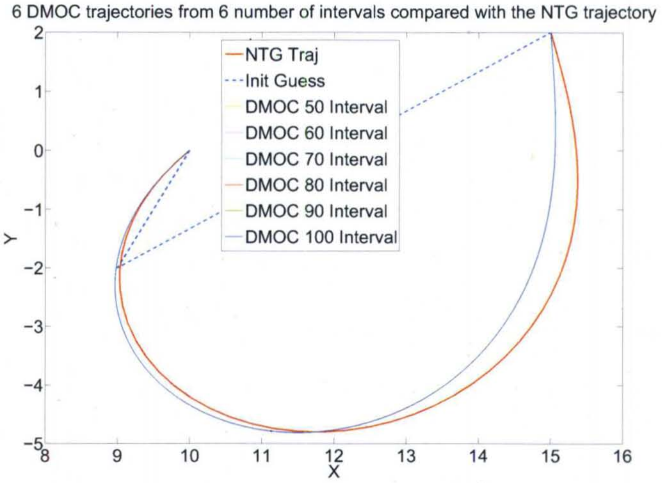


FIGURE 49 – Trajectories from DMOC versus the ones from NTG when intervals are changing.

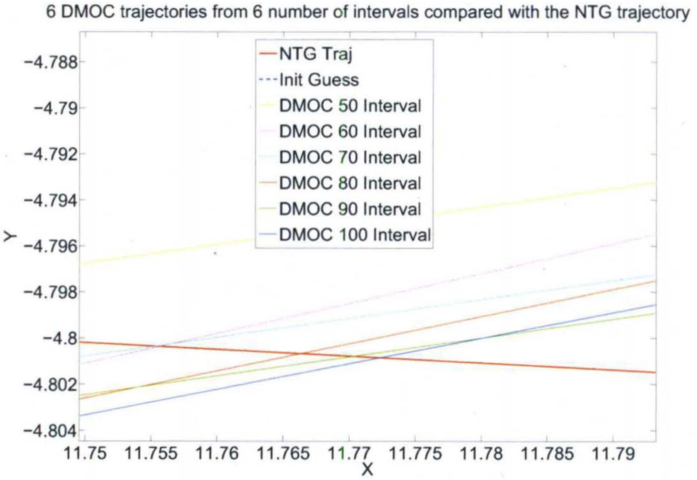


FIGURE 50 – Trajectories from DMOC in details when intervals are changing.

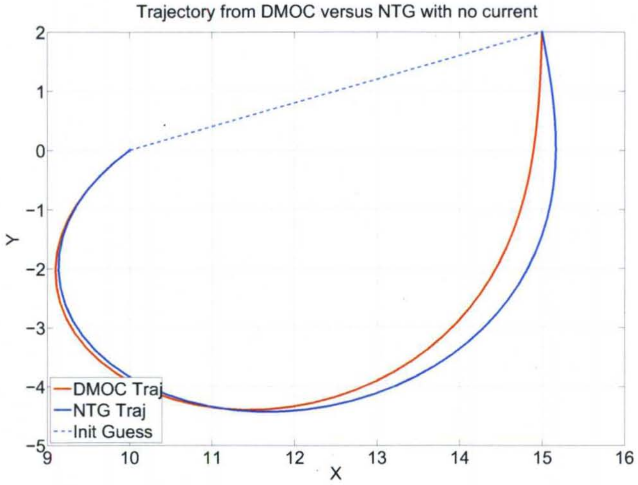


FIGURE 51 – Trajectories from DMOC versus NTG with no current.

put into these two constraints, the EL equation in x direction value is from -0.0018 to 0.0010, the average value is $-1.8753e-4$, the EL equation in y direction value is from $-6.6789e-4$ to 0.0020, the average value is $6.1872e-4$.

Further, in order to make a fair comparison, simply we specify the current as (0,0) in x and y direction. The trajectories from NTG and DMOC are shown in the following. The energy cost is 3304.243 for NTG, 3803.7 for DMOC.

The solutions from NTG and DMOC are different, it is basically due to the difference of constraints definition. NTG constraints are enforced with variables defined as B-spline variables and enforced along the collocation points while DMOC constraints are discretized and enforced on the discrete points.

When optimal controls from DMOC and NTG are fixed, the equations of motion of the glider are derived. According to the optimal controls from DMOC and NTG, new trajectories can be generated by using Matlab original differential equation solver ODE45. Their ODE45 solutions both for DMOC and NTG cannot reach the destination, and ODE 45 solution when DMOC optimal controls are applied is more closer to the original solutions. The possible reason is that the ODE45 solver is discretized, more similar as the way DMOC works while NTG is using B-spline variables which are not closely related with the variable definitions

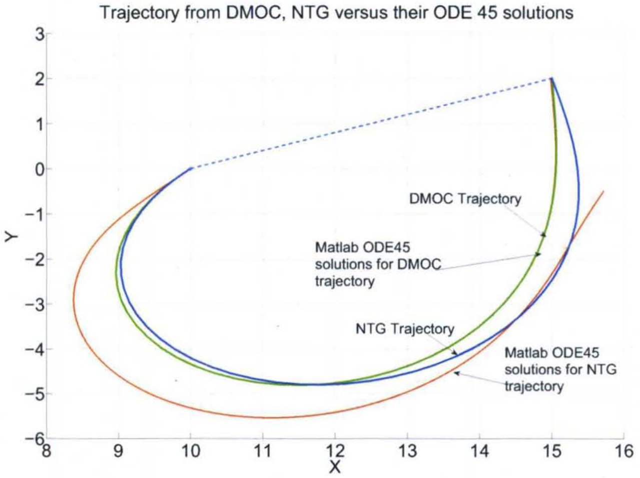


FIGURE 52 – Trajectories from DMOC and NTG versus Matlab ODE45 solutions.

in Matlab. In Figure 52, the Matlab solutions are obtained by fixing the optimal controls from DMOC and NTG, then using ODE45 solver to obtain the solutions from the original differential equations.

When the final velocity is constrained to $(0,0)$ in DMOC, nonlinear constraints are infeasible, and the trajectory is not smooth, it is shown in the following.

On the other hand, when the final velocity in NTG is fixed to $(0,0)$, NTG cannot generate good solutions either. Therefore, both DMOC and NTG cannot generate suitable solutions when the final velocity is fixed to $(0,0)$, the reason may be that in this example, the trajectory cannot be arbitrarily controlled especially at the end point since the trajectory is controlled by the gyroscopic forces. It can be proved by the following:

$$\dot{x} = V\cos\theta + u\dot{y} = V\sin\theta + v$$

Therefore

$$(\dot{x} - u)^2 + (\dot{y} - v)^2 = V^2 \tag{96}$$

It is independent of θ, while the control input τ in the gyroscopic force 79 can only control $\frac{d\theta}{dt}$. Thus the final velocity cannot be arbitrarily controlled by this control

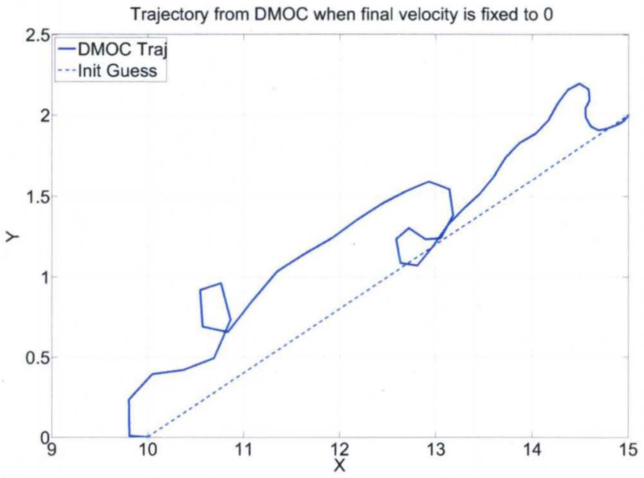


FIGURE 53 – Trajectory from DMOC when the final velocity is fixed as 0.

input.

D. A Glider in the B-spline Ocean Model

Next the characteristics of NTG and DMOC are further investigated with the dynamical glider in a more *complex environment*. The real ocean current [8] is modeled by B-spline functions shown in Figure 54 and Figure 55. The figures show the ocean current velocities at a specific time (t=13) in q_1 and q_2 directions respectively. This dynamical glider (78) is simulated moving in Monterey Bay, CA from $(-122.1780, 36.8557)$ to $(-122.2420, 36.6535)$ which represents the position in degrees (Longitude and Latitude), and it is controlled by the gyroscopic forces given in (79). The problem considered here is to find an optimal trajectory for the glider to travel from a start point to a final destination with minimum energy by taking advantage of the ocean current velocities.

In the time-varying 2D B-spline ocean current model, with different initial guesses, the glider trajectories generated from DMOC and NTG are shown in Figure 56 and Figure 57.

Trajectories from DMOC and NTG are both optimal solutions when the initial guesses are straight lines ($Guess\ 2$ in the table and figures). When initial

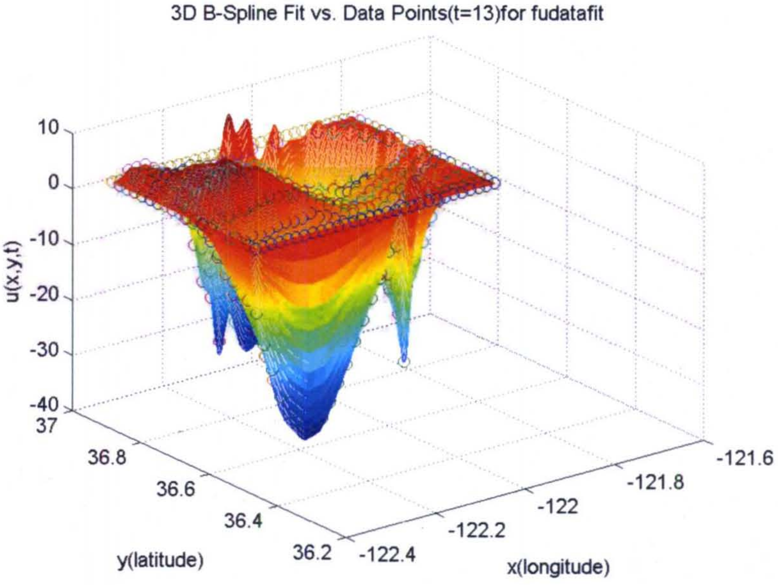


FIGURE 54 – The 3D ocean current B-spline model u(x,y,t) at t=13 hour [8].

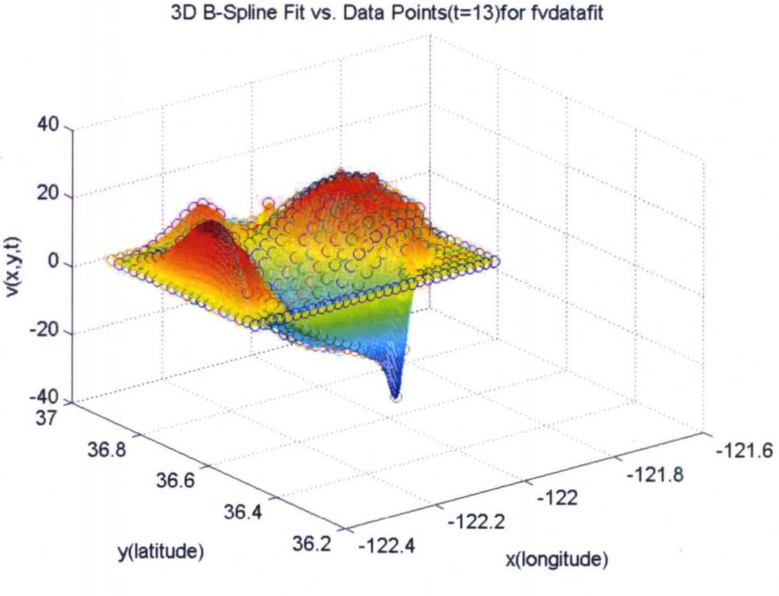


FIGURE 55 – The 3D ocean current B-spline model v(x,y,t) at t=13 hour [8].

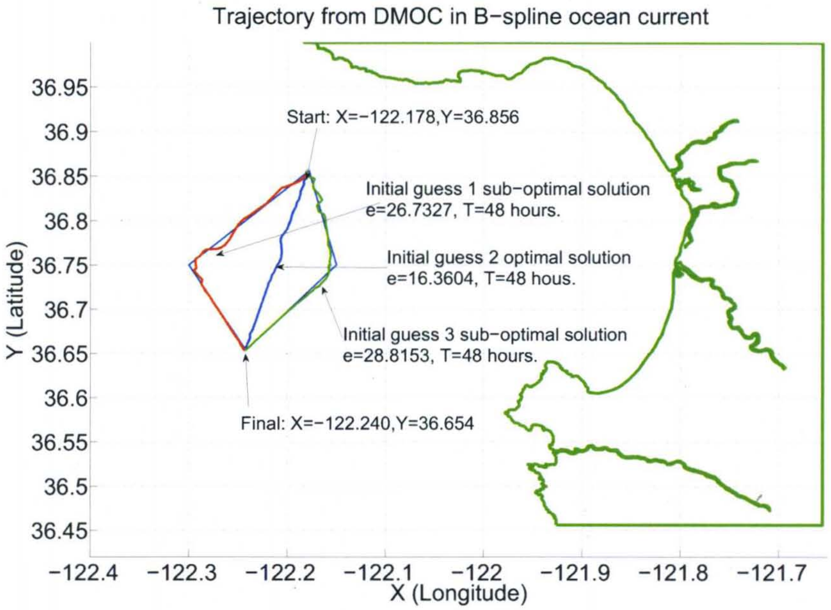


FIGURE 56 – DMOC solutions in the 3D ocean current B-spline model.

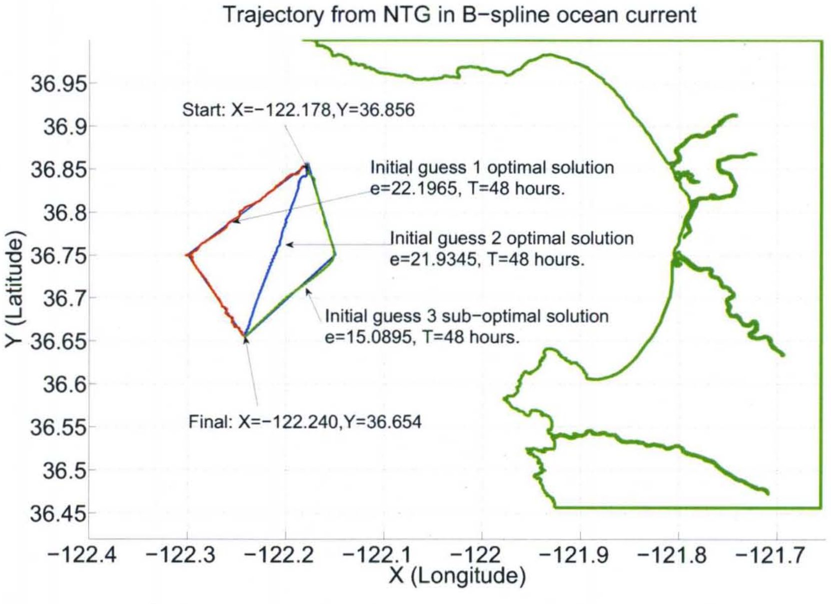


FIGURE 57 – NTG solutions in the 3D ocean current B-spline model.

TABLE 11
DMOC VS. NTG FOR A GLIDER IN A B-SPLINE OCEAN MODEL

$DMOC/NTG$	$Interval$	$T(hours)$	$Iter$	$Time(s)$	$Energy\ Cost$
$Guess\ 1$	50/50	48/48	158/25	18.11/10.98	26.7372/22.1965*
$Guess\ 2$	50/50	48/48	80/31	9.89/10.44	16.3604*/21.9345*
$Guess\ 3$	50/50	48/48	35/16	4.71/6.14	28.8153/15.0895

guesses are defined on the left and right of the straight line, DMOC and NTG only generate sub-optimal solutions. The results for comparisons are presented in TABLE 11. In TABLE 11, $Guess\ 1, Guess\ 2, Guess\ 3$ are three initial guesses which represent the guess on the left, center, right of the straight line as shown in Figure 56 and Figure 57. As it is in TABLE 7, the data with note '$*$' on the right are the optimal solutions, other data are the sub-optimal solutions. It is suitable to compare DMOC and NTG when they are both generating optimal solutions when the initial guess is a straight line. For a glider trajectory generation problem in the complex ocean current model, *it is still shown that DMOC solution costs less energy and less computation time than NTG*. It is clear for both DMOC and NTG to generate a local optimal solution, it is important to choose a suitable initial guess.

E. Hovercraft Example

Further, a hovercraft shown in Figure 58as an example [9] [83] is presented to investigate and compare DMOC with NTG, since we tried to make the problem with the constraints both as locations and velocities for the start and destination points. In this hovercraft example, the initial velocity and final velocity are both zero, it just need to fly from one location to another location with the objective to minimize the control input. The hovercraft has three degrees of freedom: its position (x, y) and its orientation θ. It has two control forces f_1 and f_2 as shown in Figure 58. They are applied at a distance r from the center of mass with f_1 acting in the direction of motion of the body and f_2 acting orthogonally to the f_1. The

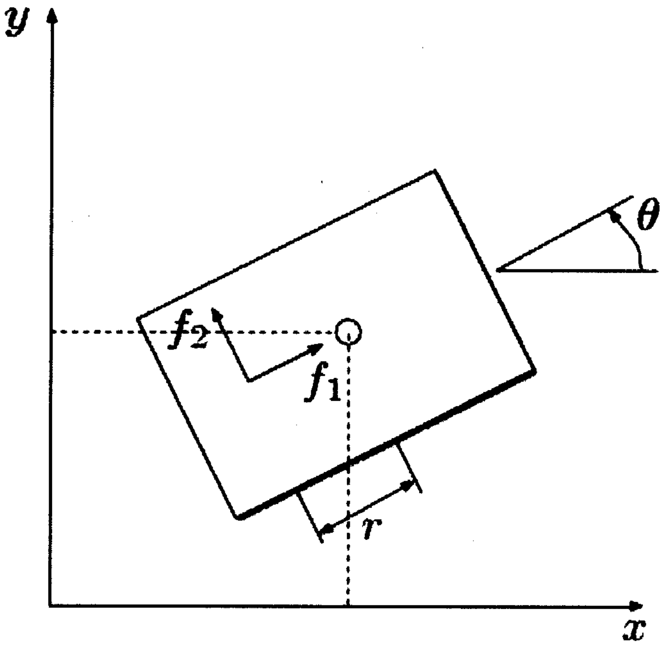


FIGURE 58 – Hovercraft [9]

Lagrangian of the system is shown as:

$$L(q,\dot{q}) = \frac{1}{2}(m\dot{x}^2 + m\dot{y}^2 + J\dot{\theta}^2) \tag{97}$$

where $q = (x, y, \theta)$, m is the mass of the hovercraft and J is the moment of inertia. The forces acting in the three directions are shown as:

$$f(t) = \begin{pmatrix} f_x = \cos\theta(t) f_1(t) - \sin\theta(t) f_2(t) \\ f_y = \sin\theta(t) f_1(t) + \cos\theta(t) f_2(t) \\ f_\theta = -r f_2(t) \end{pmatrix} \tag{98}$$

The resulted forced discrete Euler-Lagrange equations are

$$\frac{1}{M}(-q_{k-1} + 2q_k - q_{k+1}) + \frac{h}{2}\left(\frac{f_{k-1} + f_k}{2} + \frac{f_k + f_{k+1}}{2}\right) = 0 \tag{99}$$

where $k = 1, ..., N-1$ and

$$M = \begin{pmatrix} m & 0 & 0 \\ 0 & m & 0 \\ 0 & 0 & J \end{pmatrix}$$

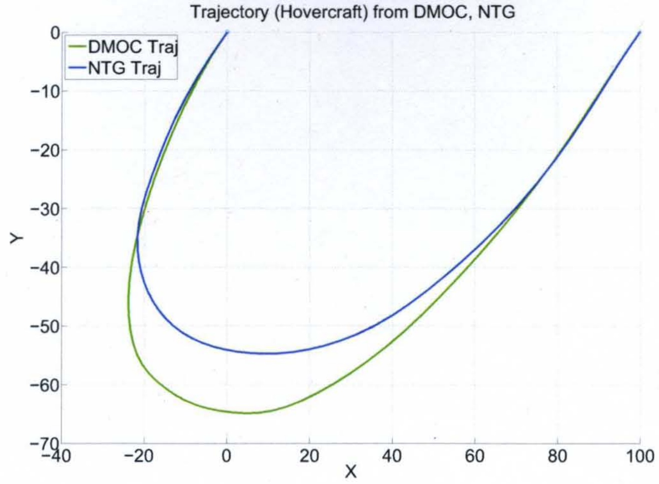


FIGURE 59 – Hovercraft trajectory from DMOC and NTG.

The goal for controlling the hovercraft is to minimize the control input f_1 and f_2, the cost function is defined as

$$J(q, f) = \int_0^1 (f_1(t))^2 + (f_2(t))^2 dt \tag{100}$$

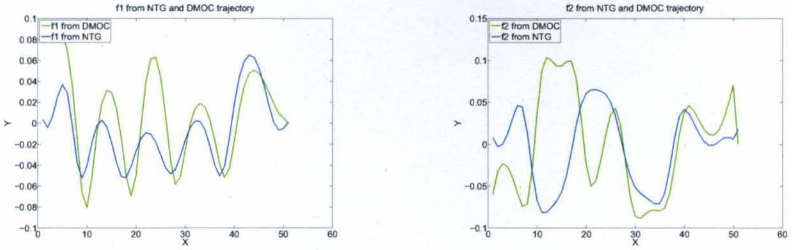
From (98) and (99), the forced discrete Euler-Lagrange equations in the x, y and θ directions are shown in the following:

$$\begin{aligned}
&\frac{1}{h}m(-x(k-1) + 2x(k) - x(k+1)) + \frac{h}{2}(\frac{f_x(k-1) + f_x(k)}{2} + \frac{f_x(k) + f_x(k+1)}{2}) \\
&\frac{1}{h}m(-y(k-1) + 2y(k) - y(k+1)) + \frac{h}{2}(\frac{f_y(k-1) + f_y(k)}{2} + \frac{f_y(k) + f_y(k+1)}{2}) \\
&\frac{1}{h}J(-\theta(k-1) + 2\theta(k) - \theta(k+1)) + \frac{h}{2}(\frac{f_\theta(k-1) + f_\theta(k)}{2} + \frac{f_\theta(k) + f_\theta(k+1)}{2})
\end{aligned} \tag{101}$$

The boundary conditions are defined as the start point $(0, 0, 0)$ and the final point as $(100, 0, pi)$. In DMOC, these conditions need to be fitted into the equations listed as 90 and 91. Their initial velocity is $(-2. -2, 0)$, the final velocity is $(2, 2, 0)$.

Both for DMOC and NTG, when the initial guess is a straight line. Their trajectories are different shown in the following. From Figure 59, DMOC trajectory costs much less computation time than NTG trajectory and comparable energy. There control forces f_1 and f_2 are shown in the following:

Further when NTG trajectory is input as an initial guess to DMOC, the



a) The control input f_1.　　b)The control input f_2.

FIGURE 60 – The control input for the trajectories shown in Figure 59.

TABLE 12
DMOC VS. NTG FOR A HOVERCRAFT TRAJECTORY

$DMOC/NTG$	$Interval$	$T(seconds)$	$Iter$	$Time(s)$	$Energy\ Cost$
$Straight\ guess$	$50/50$	$100/100$	$480/346$	$40.16/398.35$	$0.2845/0.2003$

DMOC trajectory does not change. On the other hand, the NTG trajectory does not change when the initial guess is set as the DMOC solution.

From the hovercraft example, the trajectories from DMOC and NTG are different, They generate similar trajectory and cost similar energy. The difference should be due to the numerical accuray for modelling the problem differently in DMOC and NTG. Two obvious benefits of DMOC are less compuation time and easy to model the problem as they are also shown in the glider example.

Further, more intervals are specified in DMOC to make sure that DMOC can generate a more accurate solution, however, with the increase of interval numbers, the computation time is speeding up as $O(n)$. The trajectories of the hovercraft from DMOC do not change much even the computation time is increased by 10 times as the interval number increases by 2 times.

F. Summary

In this chapter, DMOC and NTG as two different state-of-the-art optimal trajectory generation methods are compared. In application, NTG is more difficult than DMOC to be applied since several variables have to be defined as B-spline functions, in which parameters should be suitably chosen. On the other hand, there is only one DMOC parameter which can be adjusted to generate an optimal (sub-optimal) solution. These two methods are analyzed and compared with optimal trajectory generating problems of a dynamical glider both in a simplified and a complex ocean current model. For a simplified ocean current model, with the right selections of NTG parameters, the glider optimal trajectories from DMOC and NTG are similar as shown in Figure 45 through Figure 48. DMOC can generate an optimal solution which costs comparable energy and less computation time than NTG sub-optimal solution. Furthermore, as shown in TABLE 11, in a complex B-spline ocean current model, when DMOC and NTG both generate similar optimal solutions, DMOC still costs comparable energy and less computation time. In addition, a hovercraft example further shows that DMOC can save much computation time to generate similar optimal solutions to the NTG, the difference between DMOC trajectory and NTG trajectory is related to the numerical reasons. In summary, this chapter shows that DMOC can generate optimal solutions with comparable energy consumption and less computation time than NTG.

CHAPTER VI

THE UOFL MARIT TESTBED

A. The MARIT Testbed

A MARIT (Multiple Air Robotics Indoor Testbed) testbed is established at the University of Louisville for investigating control algorithms, it is upgraded from the previous mobile robot testbed [84]. The system consists of a Vicon 8i motion capture system and draganflyer helicopters on which markers are attached, therefore they can be tracked by 6 Vicon Mcam 2 cameras. The similar testbeds exist in Vanderbelt [85], MIT [86], and University of Essex [87], United Kingdom. This testbed further can be used to study autonomous systems in which several agents cooperate with each other to perform some assigned tasks. UAVs are attracting quite a lot of attention shown in [88] [89] [90] since UAVs provide convenient and cost-effective tools for various applications [91] including terrain and utilities monitoring or environmental surveillance, search and rescue, aerial mapping, traffic surveillance. Benefits of indoor testbed include that it can perform testing purposes regardless of outside weather conditions and easy to monitor and control.

B. Vicon Vision System

Vicon motion capture systems have been used in life science, sports, medical, movie and game industry, music, robot to accurately track and analyze movements. The testbed established at U of L is utilizing a Vicon system to track helicopters, then to study control methods for helicopters. The system consists of 6

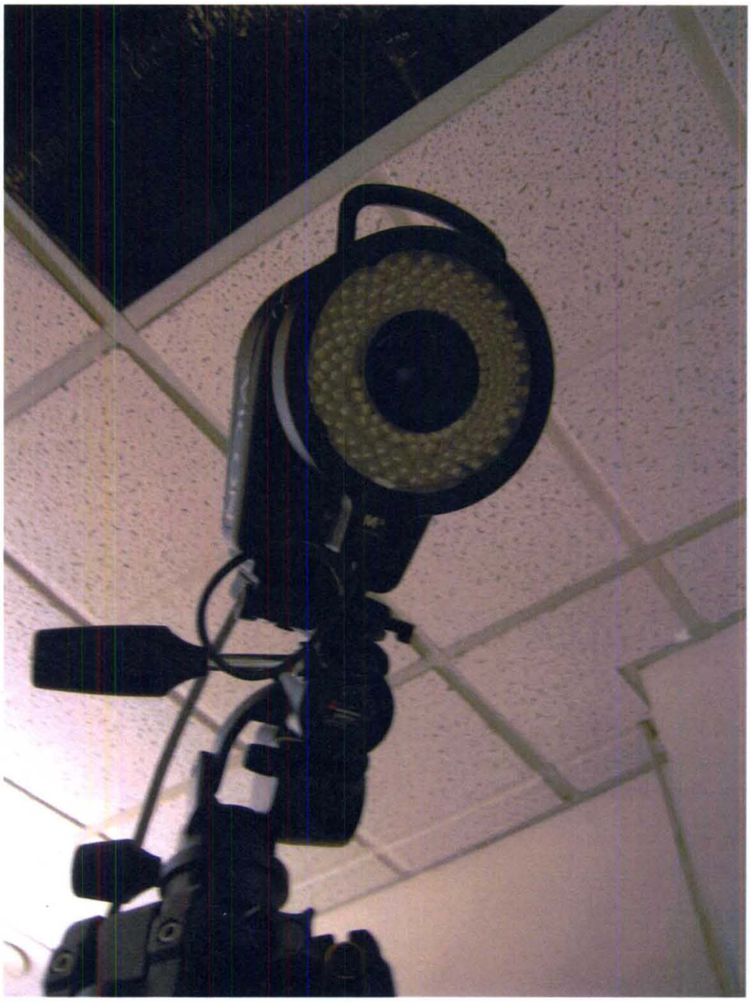


FIGURE 61 – A Mcam 2 camera in the Vicon Motion Capture System.

Mcam 2 cameras shown in Figure 61, one datastation presented in Figure 62, two third party megapixel Camera Interface Unit shown in Figure 63, several video channel distribution cables, one workstation shown in Figure 64 which is a dedicated desktop with Vicon iQ 2.0 in Windows XP. The datastation is connected to the cameras through camera interface units with every three cameras using an Unit. Also, the datastation is connected with the workstation with a crossover cable.

C. Real-Time Application

In order to make the Vicon system obtain real-time 6 degrees of freedom in-

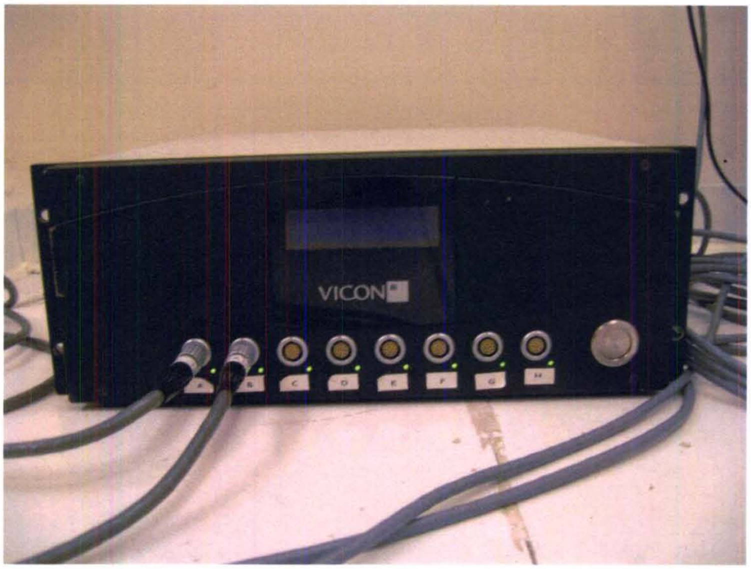


FIGURE 62 – The datastation in the Vicon Motion Capture System.

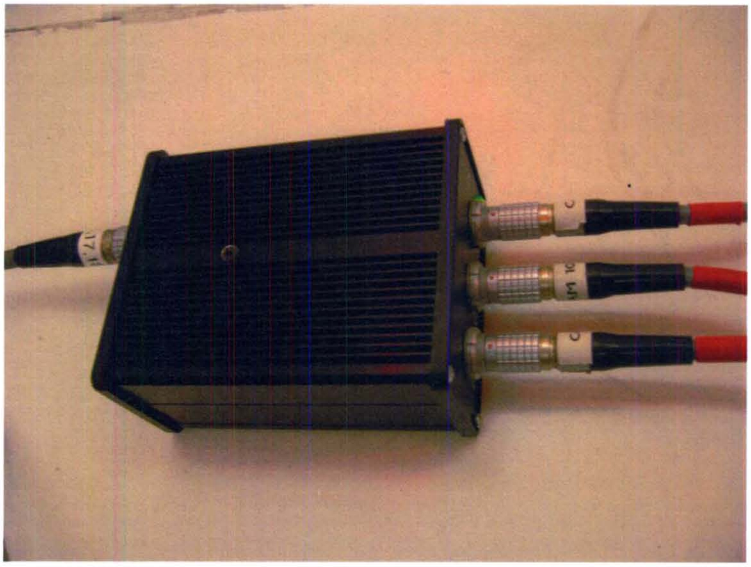


FIGURE 63 – A camera interface unit in the Vicon Motion Capture System.

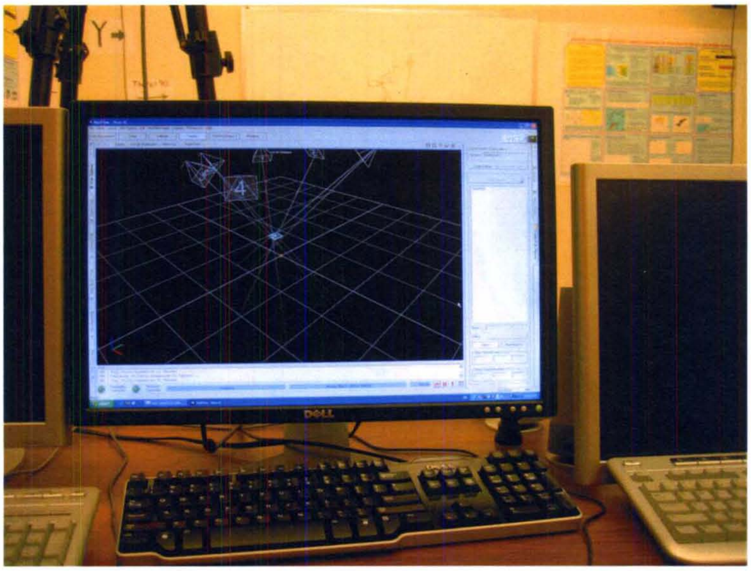


FIGURE 64 – The workstation in the Vicon Motion Capture System.

formation, first the system should be calibrated, the calibration steps can be found in the Vicon Manual [92]. After calibration, a program is written in C++ with the application of Vicon Real Time SDK. The following steps are done to achieve the task.

- 1, set up 6 vicon cameras and calibrate these cameras so that the draganflyer can be checked with 5 markers on it
- 2, in Vicon IQ, create a rigid body with 5 markers, the rigid body is named as draganflyer.
- 3, connect the workstation with the datastation by specifying the proper IP addresses.
- 4, download Vicon Real Time SDK.
- 5, in Visual C++, create a project, include VrtSDK10ex.h header files in your folder. link with VrtSDK10ex.lib. In Visual c++, Project− >Add Existing Item− > choose VrtSDK10ex.lib in thefolder).
- 6, begin to write a program, our program is shown in the attachment

After real-time coordinates of the draganflyer are obtained, the future step is to send the comands to the draganflyer in real-time through the computer. Since

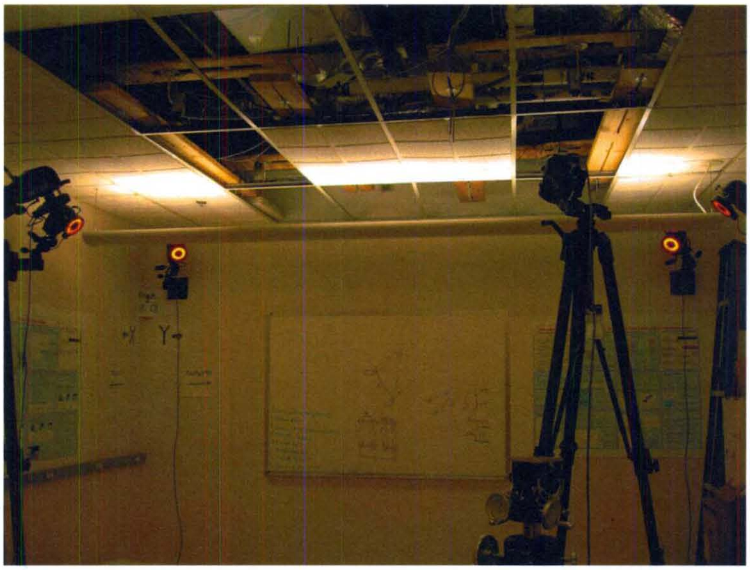


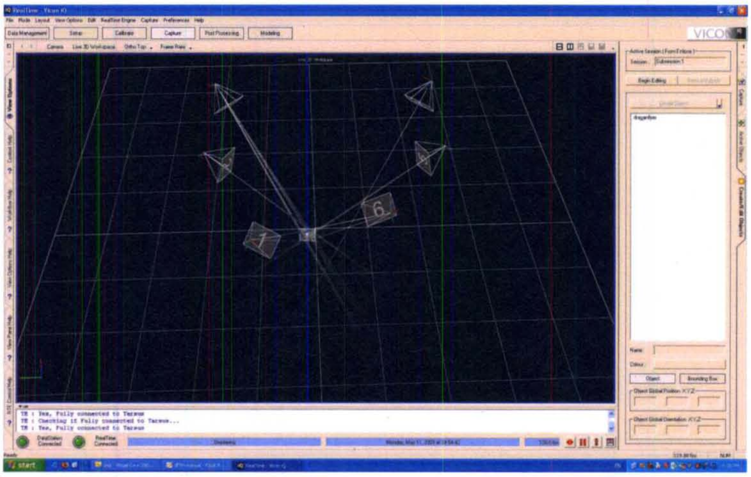
FIGURE 65 – The Vicon Motion Capture System.

the cameras are already calibrated, the 6 cameras are shown in the 3D live space as Figure 66. For the tracked draganflyer, five markers are attached on the body. These five markers shown in Figure 67 are defined as a rigid body, thus its center coordinates and its orientation angles can be obtained by the program with the application of Vicon Real-Time SDK.

In the C++ program, the real-time coordinates and orientation angles can be obtained as shown in Figure 68.

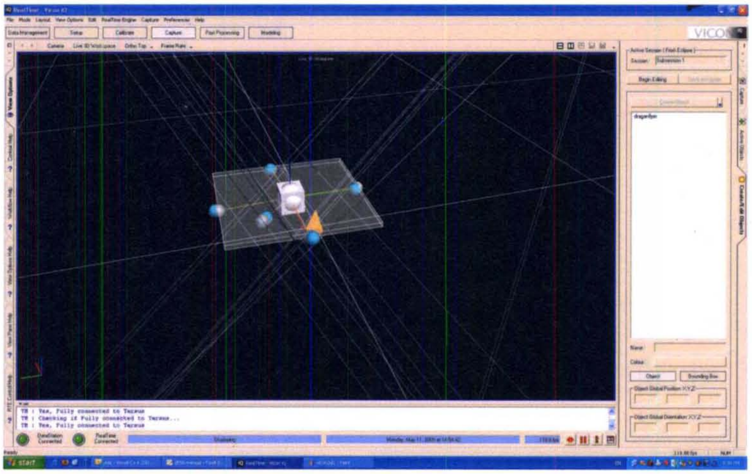
D. Summary

The UAV 3D testbed is established with Vicon 8i motion capture system. 6 cameras connected with the datastation are able to track real-time coordinates of any suitable vehicle or helicopter, even biological activities. The Vicon system should first be calibrated, then a c++ program should be written by the user to process the raw data from the Vicon system after the program is connected with Vicon Real-Time Engine. After setting up the Vicon System, the draganflyer helicopter is modelled as a rigid body, its 6 DOF information can be obtained in real-time



```
TR : Yes, Fully connected to Tarsus
TR : Checking if Fully connected to Tarsus...
TR : Yes, Fully connected to Tarsus
```

FIGURE 66 – The Vicon iQ 2.0 in 3D live work space.



```
TR : Yes, Fully connected to Tarsus
TR : Checking if Fully connected to Tarsus...
TR : Yes, Fully connected to Tarsus
```

FIGURE 67 – The markers are tracked in the Vicon Motion Capture System.

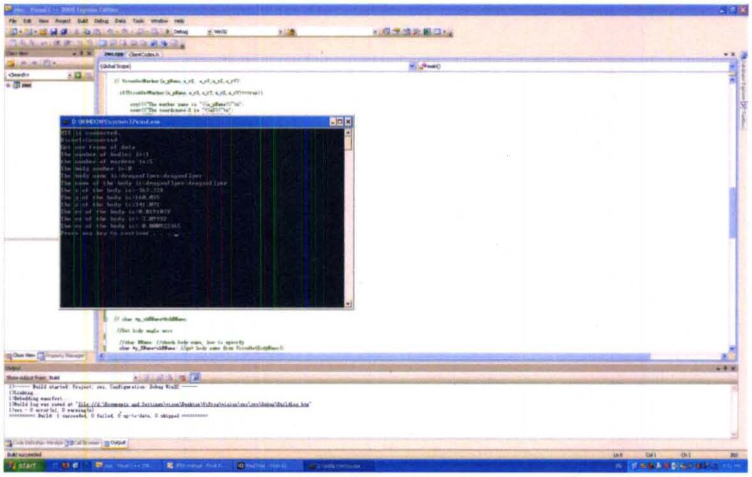


FIGURE 68 – Real-time coordinates output in the Vicon Motion Capture System.

with our own C++ program based on Vicon Real-Time SDK. This motion capture system establishes a good foundation for verifying optimal trajectory generation methods such as DMOC and NTG by experiments.

CHAPTER VII

CONCLUSION AND FUTURE WORK

A. Conclusion

In this dissertation, DMOC and NTG as two state-of-the-art optimal trajectory generation methodologies were investigated with application to an underwater glider and a JPL aerobot. These two optimal trajectory generation methods were analyzed and compared with application to a glider in both simple ocean current and B-spline current. For the detailed conclusion in the every chapter, they are listed as follows:

In Chapter I, the motivation for this research was presented which indicated that robotic explorers for ocean and outer space will be necessary tools to discovery and advancements in science and technology. In order to make robotic rovers explore the unknown places efficiently and robustly, optimal control methodologies need to be utilized. Then two state-of-the-art trajectory generation methodologies were introduced, NTG and DMOC. Finally, outline of this dissertation was presented in this chapter.

Chapter II presented optimal trajectories from NTG for an underwater glider to strengthening the previous hypodissertation that LCS (Lagrangian Coherent Strutures) in the ocean reveal efficient or near-optimal routes for glider transport. In this chapter, with modelling the glider kinematically and dynamically, trajectories found with the 3D B-spline ocean flows model corresponds well with LCS, for which numerical solutions of several scenarios and animations of glider trajectories with Tecplot were presented.

Chapter III proposed to utilize Nonlinear Trajectory Generation methodology to generate 3D opportunistic trajectories for an aerobot by utilizing wind information. The aerobot is dynamically controlled by three propellers which are respectively parallel to the local three Cartesian axes. Constraints for the aerobot control are derived from Euler-Lagrange equations in the condition that the aerobot must be satisfied with the Lagrange-D'Alembert principle. The new proposed aerobot model takes the aerodynamics into account. The results show that NTG can take the advantage of wind profiles to save significant energy for the defined goal. Further, a state space model of the Aerobot which decoupled its longitudinal and lateral dynamics is also investigated to generate the optimal trajectories. The minimizing energy trajectory with this complex model did cost more time than the simple model but the optimal trajectory is still energy efficient with NTG.

Chapter IV presented a detailed procedure to apply DMOC methodology to solve optimal control problems. It explains the principle of DMOC, and how to formulate the problem in DMOC. Then the steps are shown about how to install and configure nonlinear programming solver IPOPT, and how to use the modeling language AMPL. In particular, a user-defined function is involved with AMPL to solve more complicated problem.

In Chapter V theoretical foundations and results for comparisons were presented with application to a dynamic glider. In a simple ocean current example and a B-spline ocrean current model, DMOC optimal solutions are shown to easier to generate, cost less computation time and comparable energy than NTG optimal solutions.

Chapter VI presented the MARIT testbed with Vicon 8i motion capture system. The real-time 6 DOF information of a defined rigid body can be tracked. The testbed is being established for future research.

B. Future Work

Since the MARIT testbed is being established and it is able to get real-time coordinates of an object which can be a mobile robot or a draganflyer helicopter, future research is focused on implementing and validating NTG and DMOC in the control of draganflyers. NTG and DMOC can be embedded in the program which gets the position information from Vicon system, then the program can generate trajectory which consists of a few of waypoints with NTG or DMOC. Based on the reference trajectory given by NTG or DMOC, the program can send the commands wirelessly to the draganflyers to make them fly from one start point to one destination point by specified routes minimizing energy, and or minimizing time.

REFERENCES

[1] The NOAA posts new updates to Jacksonville area nautical chart, safe navigation important to Super Bowl vistors. `www.noaanews.noaa.gov/stories2005/s2383.htm`, Feb 2005.

[2] Mars exploration rovers. `http://www.nasa.gov/centers/jpl/missions/mer.html`.

[3] Mars aerobot image. `http://www.fourth-millennium.net`.

[4] H. Stommel. The slocum mission. *Oceanography*, 2:22–25, 1989.

[5] Nasa-jpl planetary aerovehicles. `http://www.jpl.nasa.gov`.

[6] V. R. Cortés, J. R. Azinheira, and E. C. Paiva. Identification of lateral dynamics of aurora airship. 2004.

[7] A. Wăchter. *Introduction to IPOPT: A Tutorial for Downloading, Installing, and Using IPOPT*. 2008.

[8] Weizhong Zhang, Tamer Inanc, S. O. baum, and Jerrold E. Marsden. Optimal trajectory generation for a dynamic glider in ocean flows modeled by 3d b-spline functions. In *Proceedings of ICRA 2008*, CA, May 2008.

[9] O. Junge, J. E. Marsden, and S. Ober-blőbaum. Discrete mechanics and optimal control. In *Proccedings of the 16th IFAC World Congress*, page 6, 2005.

[10] Autonomous ocean sampling network homepage. `http://www.mbari.org/aosn/`, Jun 2006.

[11] T. Inanc, S. C. Shadden, and J. E. Marsden. Optimal trajectory generation in ocean flows. In *Proceedings of 2005 American Control Conference*, Jun 2005.

[12] F. Zhang, D. M. Fratantoni, D. Paley, J. Lund, and N. E. Leonard. Control of coordinated patterns for ocean sampling. *International Journal of Control, special issue on Navigation, Guidance and Control of Uninhabited Underwater Vehicles*, 80(7):1186–1199, July 2007.

[13] P. Bhatta, E. Fiorelli, F. Lekien, and N. E. Leonard et al. Coordination of an underwater glider fleet for adaptive ocean sampling. In *Proc. International Workshop on Underwater Robotics, Int. Advanced Robotics Programmed (IARP)*, 2005.

[14] J. G. Graver, R. Bachmayer, N. E. Leonard, and D. M. Fratantoni. Underwater glider model parameter identification. In *Proc. 13th Int. Symp. on Unmanned Untethered Submersible Technology (UUST)*, Aug 2003.

[15] J. Sherman, R. E. Davis, W. B. Owens, and J. Valdes. The autonomous underwater glider spray. *IEEE J.Oceanic Engin.*, 26(4):437–446, 2001.

[16] J. J. Fahie. *Galileo: His Life and Work.* Adamant Media Corporation, 2000.

[17] L. P. Gerson. *Aristotle and Other Platonist.* Ithaca : Cornell University Press, 2005.

[18] W. D. Compton. *Where No Man Has Gone Before: A History of Apollo Lunar Exploration Missions.* DIANE Publishing Co, 1996.

[19] Draganfly innovations inc. `http://www.rctoys.com`.

[20] H. J. Sussmann and J. C. Willems. 300 years of optimal control: from the brachystochrone to the maximum principle. *Control Systems Magazine, IEEE*, 17(3):32–44, 1997.

[21] A. E. Bryson. Optimal control-1950 to 1985. *Control Systems Magazine, IEEE*, 16(3):26–33, 1996.

[22] L. S. Pontryagin, V. G. Boltyanskii, R. V. Gamkrelidze, and E. F. Mishchenko. *The Mathematical Theory of Optimal Processes.* Interscience publishers, a division of John Wiley & Sons, Inc., 1962.

[23] R. Bellman. *Dynamic Programming.* Princeton University Press, 1957.

[24] C. R. Hargraves and S. W. Paris. Direct trajectory optimization using nonlinear programming and collocation. *J. Guidance, Control and Dynamics*, 10(4), 1987.

[25] J. E. Mcinroy and J. A. Mellstrom. Optimal trajectory generation for linear systems. In *American Control Conference, 1995. Proceedings of the*, volume 5, pages 3121–3125, 1995.

[26] A. Fabbrini, D. Doretti, S. Braune, A. Garulli, and P. Mercorelli. Optimal trajectory generation for camless internal combustion engine valve control. In *Industrial Electronics, 2008. IECON 2008. 34th Annual Conference of IEEE*, pages 303–308, 2008.

[27] D. Pekarek, A. D. Ames, and J. E. Marsden. Discrete mechanics and optimal control applied to the compass gait biped. In *Proc. of the 46th IEEE Conf. on Decision and Control*, pages 5376–5382, Dec 2007.

[28] C. Xu, A. Ming, and M. Shimojo. Optimal trajectory generation for manipulator with strong nonlinear constraints and multiple boundary conditions. In *Robotics and Biomimetics, 2004. ROBIO 2004. IEEE International Conference on*, pages 192–197, 2004.

[29] M. Guilbert, P. Wieber, and L. Joly. Optimal trajectory generation for manipulator robots under thermal constraints. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 742–747, 2006.

[30] B. Mcavoy, B. Sangolola, and Z. Szabad. Optimal trajectory generation for redundant planar manipulators. In *Systems, Man, and Cybernetics, 2000 IEEE International Conference on*, volume 5, pages 3241–3246 vol.5, 2000.

[31] B. H. Lee, J. S. Kong, and J. G. Kim. Optimal trajectory generation for a humanoid robot based on fuzzy and genetic algorithm. In *2006 IEEE Congress on Evolutionary Computation*, pages 1968–1974, 2006.

[32] E. Arnold, J. Neupert, O. Sawodny, and K. Schneider. Trajectory tracking for boom cranes based on nonlinear control and optimal trajectory generation. In *Proceedings of the 41st IEEE Conf. on Decision and Control*, Las Vegas, Dec.

[33] P. D. Hattis and R. K. Smolskis. Optimal trajectory generation and design trades for hypersonic vehicles. In *American Control Conference, 1989*, pages 1125–1130, 1989.

[34] C. D. Boor. *A Practical Guide to Splines*. Springer, 2003.

[35] A. Elfes, J. L. Hall, J. F. Montgomery, C. F. Bergh, and B. A. Dudik. Towards a substantially autonomous aerobot for exploration of titan. In *in the Proceedings of the 2004 IEEE International Conference on Robotics & Automation, New Orleans, LA,*, pages 2535–2541, Apr 2004.

[36] M. Fox and D. Long. Single-trajectory opportunistic planning under uncertainty. In *Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space*, 2002.

[37] D. Johnstone and S. Bradley. Opportunistic scheduling in a constraint-rich world. *SIGBED Rev.*, 2(2):19–22, 2005.

[38] M. B. Milam, K.Mushambi, and R. M. Murray. A new computational approach to real-time trajectory generation for constrained mechanical systems. In *Proc. of IEEE Conference on Decision and Control*, pages 845–851, 2000.

[39] V. Jurdjevic. *Geometric Control Theory*. Cambridge University Press, 1997.

[40] Michel Fliess, Jean Lévine, and Pierre Rouchon. Flatness and defect of nonlinear systems: Introductory theory and examples. *International Journal of Control*, 61:1327–1361, 1995.

[41] P. T. Boggs and J. W. Tolle. Sequential quadratic programming. *Acta Numerica*, pages 1–50, 1995.

[42] Jr. A. E. Bryson and Y. C. Ho. *Applied Optimal Control: Optimization, Estimation and Control*. Washington: Hemisphere Pub. Corp., 1975.

[43] M. B. Milam. *Real time Optimal Trajectory Generation for Constrained Dynamical Systems*. 2003.

[44] P. E. Gill, W. Murray, M. Saunders, and M. Wright. Npsol nonlinear programming software.

[45] B. A. Murtagh and M. A. Saunders. *MINOS 5.5 USER'S GUIDE*, Jul 1998.

[46] P. E. Gill, S. J. Hammarling, W. Murray, M. A. Saunders, and M. H. Wright. User's guide for lssol(version 1.0): A fortran package for constrained linear least-squares and convex quadratic programming. Technical report, Jan 1986.

[47] R. Bachmayer, N. E. Leonard, J. Graver, E. Fiorelli, P. Bhatta, and D. Paley. Underwater gliders: Recent developments and future applications. In *Proc. of 2004 International Symposium on Underwater Technology*, pages 195–200, 2004.

[48] D. L. Rudnick, R. E. Davis, C. C. Eriksen, D. M. Fratantoni, and M. J. Perry. Underwater gliders for ocean research. *Marine Technology Society Journal*, 38, 2004.

[49] S. A. Jenkins, D. E. Humphreys, J. Sherman, J. Osse, C. Jones, N. Leonard, J. Graver, R. Bachmayer, T. Clem, P. Carroll, P. Davis, J. Berry, P. Worley, and J. Wasyl. Underwater glider system study. May 2003.

[50] B. Curtin, J. G. Bellingham, J. Catipovic, and D. Webb. Autonomous oceanographic sampling networks. *Oceanography*, 6:86–94, 1989.

[51] G. Haller. Lagrangian coherent structures from approximate velocity data. *Physics of Fluids*, 14(6):1851–1862, June 2002.

[52] J. D. Paduan and L. K. Rosenfeld. Remotely sensed surface currents in monterey bay from shore-based hf radar (coastal ocean dynamics application radar). *Journal of Geophysical Research*, 101:20669–20686, 1996.

[53] F. Lekien, C. Coulliette, R. Bank, and J. Marsden. Open-boundary modal analysis: Interolation, extrapolation and filtering. *Journal of Geophysical Research Oceans*, 109, 2004.

[54] J. A. Cuttes, T. S. Balint, A. P. Belz, and C. E. Peterson. Overview of nasa's 2006 sse strategic roadmap. In *Aerospace Conference, 2007 IEEE*, pages 1–10, 2007.

[55] J. Jones. Inflatable robotics for planetary applications. In *6th International Symposium on Artificial Intelligence, Robotics, and Automation in Space, Montreal, Canada*, Jun 2001.

[56] B. Kroplin. Solar airship lotte, technical report, institute for statics and dynamics of aerospace structures. Technical report, 2002.

[57] T. Kampke and A. Elfes. Optimal wind-assisted flight planning for planetary aerobots. In *Proceedings of the 2004 IEEE International Conference on Robotics & Automation, New Orleans, LA,*, pages 2542–2549, Apr 2004.

[58] J. Biesiadecki, A. Jain, and M. L. James. Advanced simulation environment for autonomous spacecraft. In *International Symposium on Artificial Intelligence, Robotics and Automotion in Space.*

[59] M. S. Hassouna and A. A. Farag. Dsends - a high-fidelity dynamics and spacecraft simulator for entry, descent and surface landing. In *Proceeding of IEEE Aerospace Conference*, volume 7, pages 3343–3359, 2002.

[60] R. M. Young. Model 81000 ultrasonic anemometer. `http://www.youngusa.com`.

[61] J. D. Anderson. *A History of Aerodynamics and its Impact on Flying Machines*. New York: Cambridge University Press, Cambridge, 1998.

[62] G. A. Khoury and J. D. Gillett. *Airship Technology*. New York: Cambridge University Press, Cambridge, 1999.

[63] V. V. Kerzhanovich and J. A. Cutts. Aerobots in planetary exploration. In *Proc. of IEEE Aerospace Conference 2000*.

[64] Alberto Elfes, James F. Montgomery, Jeffery L. Hall, Sanjay S. Joshi, Jeffrey Payne, and Charles F. Bergh. Autonomous flight control for a titan exploration aerobot. In *Proc. of the 8th International Symposium on Artificial Intelligence, Robotics and Automation in Space*, 2005.

[65] A. Elfes and José R. Azinheira. Robotic airships for exploration of planetary bodies with an atmosphere: Autonomy challenges. *Autonomous Robots*, 14:147–164, 2003.

[66] V. R. Cortés, J. R. Azinheira, E. C. Paiva, B. Faria, J. Ramos, and S. Bueno. Experimental identification of aurora airship.

[67] J. E. Marsden and M. West. Discrete mechanics and variational integrators. *Acta Numerica*, 10:357–514, 2001.

[68] S. Lall and M. West. Discrete variational hamiltonian mechanics. *Journal of Physics A: Mathematical and general*, 39:5509–5519, 2006.

[69] O. Junge, J. E. Marsden, and S. Ober-Blöbaum. Optimal reconfiguration of formation flying spacecraft–a decentralized approach. In *Proceedings of the 45th IEEE Conf. on Decision and Control*, pages 5210–5215, Dec 2006.

[70] M. Desbrun, A. N. Hirani, and J. E. Marsden. Discrete exterior calculus for variational problems in computer vision and graphics. In *Proceedings of the 42nd IEEE Conf. on Decision and Control*, Dec 2003.

[71] C. W. Rowley and J. E. Marsden. Variational integrators for degenerate lagrangians, with application to point vortices. In *Proceedings of the 41st IEEE Conf. on Decision and Control*, pages 1521–1527, Dec 2002.

[72] L.T. Biegler. Solution of dynamic optimization problems by successive quadratic programming and orthogonal collocation. Technical report, 1983.

[73] W. Zhang, T. Inanc, and J. E. Marsden. A tutorial for applying dmoc to solve optimization control problems. submitted to the 2010 American Control Conference, 2010.

[74] S. J. Wright. *Primal-dual Interior-Point Methods.* Society for Industrial and Applied Mathematics, 1997.

[75] A. Wchter and L. T. Biegler. On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming. *Mathematical Programming:Series A and B*, 106:25–57, 2006.

[76] Computational infrastructure for operations research. `https://projects.coin-or.org/Ipopt`.

[77] R. Fourer, D. M. Gay, and B. W. Kernighan. A modeling language for mathematical programming. *Manage. Sci.*, 36:519–554, 1990.

[78] R. Vanderbei. Nonlinear optimization models. `http://www.sor.princeton.edu/~rvdb/ampl/nlmodels/`.

[79] D. M. Gay. Hooking your solver to ampl. Technical report, 1997.

[80] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright. Some theoretical properties of an augmented lagrangian merit function. *Advances in Optimization and Parallel Computing*, pages 101–128, 1992.

[81] R. M. Murray, M. Rathinam, and W. Sluis. Differential flatness of mechanical control systems: A catalog of prototype systems. In *Proceedings of 1995 ASME Int'l Mech Eng Congress and Expo*, pages 1–9, November 1995.

[82] O. Junge, S. Ober-Blőbaum, and J. E. Marsden. Dmoc project. `http://www.cds.caltech.edu/~marsden/research/demos/dmoc/`.

[83] O. Junge and S. Ober Blőbaum. Optimal reconfiguration of formation flying satellites. In *Proc of IEEE Conference on Decision and Control and European Control Conference ECC*, 2005.

[84] T. Riggs, T. Inanc, and W. Zhang. The UofL autonomous mobile robotics systems testbed. *Accepted to IEEE Transactions on Control Systems Technology*, Dec 2008.

[85] T. J. Koo. Vanderbilt embedded computing platform for autonomous vehicles. `http://www.vuse.vanderbilt.edu/~kootj/Projects/VECPAV/`, Jul 2006.

[86] J. P. How, B. Bethke, A. Frank, D. Dale, and J. Vian. Real-time indoor autonomous vehicle test environment. *Control Systems Magazine, IEEE*, 28(2):51–64, 2008.

[87] John Oyekan and Huosheng Hu. Toward autonomous patrol behaviors for uavs. `http://cswww.essex.ac.uk/staff/hhu/Papers/HAM%202009.pdf`.

[88] Stanford testbed of autonomous rotorcraft for multi-agent control. `http://hybrid.eecs.berkeley.edu/index.php?section=31`.

[89] The Berkley aerial robot project. `http://robotics.eecs.berkeley.edu/bear/`.

[90] USC autonomous flying vehicle project. `http://www-robotics.usc.edu/~avatar/`.

[91] A. Ollero and L. Merino. Control and perception techniques for aerial robotics. *Annual Reviews in Control*, 28:167–178, 2004.

[92] Vicon company. `http://www.vicon.com/`.

APPENDIX I

NOMENCLATURE

The following convention is used throughout this dissertation.

NTG	Nonlinear Trajectory Generation
DMOC	Discrete Mechanics and Optimal Control
SQP	Sequential Quadratic Progromming
LCS	Lagrangian Coherent Structure
NASA	National Aeronautics and Space Administration
JPL	Jet Propulsion Laboratory
AMPL	A Mathematical Programming Language
IPOPT	Interior Pointer OPTimizer
UAV	Unmanned Air Vehicle
PCTx	PC to Transmitter Interface
$u(x, y)$	2D ocean current velocity in x direction
$v(x, y)$	2D ocean current velocity in y direction
$B_{i,k}$	B-spline basis function for the x direction
$B_{j,k}$	B-spline basis function for the y direction
$u(x, y, t)$	3D ocean current velocity in x direction
$v(x, y, t)$	3D ocean current velocity in y direction
F_{gyr}	Gyroscopic Force
$q(t)$	System State
$L(q(t), \dot{q}(t))$	Lagrangian of the system
Q	Configuration Space
h	Step Size of Discretization
$L_d(q_k, q_{k+1})$	Discrete Lagrangian
$J_d(q_d, f_d)$	Discrete Cost Function

APPENDIX II

NTG program for a glider in a B-spline ocean model

```
//The following NTG Program is to generate trajectories for a glider in a B-spline Ocean Model.
//The programs have four parts. They are opt1.main.c which is the main program, opt1.sub.h is the B-spline
//ocean current model program, opt1.inp which is the input file for the NTG, opt1.make which is makefile
//for the program.
// opt1.main.c
#include <stdlib.h>
#include <math.h>/* math functions */
#include <ntg.h>/* main NTG declarations */
#include <time.h>       /*get the time*/
#include <ParseInputFile.c>
int main(int argc, char *argv[])
{
OPTPARAM optparam;
int i,j,sum, k1;
char *fname;
FILE *fp;
FILE *fp2;
FILE *fcurrent;
FILE *fcurrent2;
double **Traj,*Time;
int nTraj, Traj_offset,coef_offset, nPts=500;
float currentdatau[4] = {0};
float currentdatav[4] = {0};
double Tf = 0;
FILE *fxinit, *fyinit;
const int sizeofinit = 93;
float xinit[sizeofinit];
float yinit[sizeofinit];
float xstart, xstop, xdif, ystart, ystop, ydif;
double **knots; /* knot points, list of times for each output */
int nbps;
double *bps;
double **lic,**lfc, **ltc;

/* initial guess size = sum over each output ninterv*(order-mult)+mult */
int ncoef;
int *NCOEF;
double *coefficients;
int *istate;
double *clambda;
double *R;
int inform;
double objective;
if (argc!=2){
printf("\n\tUsage: %s inputfile.inp\n\n",argv[0]);
exit(-1);

}
```

```
//Read Input file
parse_input_file(argv[1],&optparam);
// Allocate space and initialize the knot points
knots=(double **) malloc(optparam.nout*sizeof(double*));
for (i=0;i<optparam.nout;i++){
knots[i]=(double *) malloc((optparam.ninterv[i]+1)*sizeof(double));
linspace(knots[i], 0, optparam.HL, optparam.ninterv[i]+1);
}
ncoef = 0;
NCOEF = (int*) malloc(optparam.nout*sizeof(int));
for(i=0;i<optparam.nout;i++){
ncoef = ncoef + (optparam.ninterv[i]*(optparam.order[i]-optparam.mult[i])
+optparam.mult[i]);
NCOEF[i] = optparam.ninterv[i]*(optparam.order[i]-optparam.mult[i])
+optparam.mult[i];

  }
/* Initial guess for coefficients (all 0s)*/
coefficients=(double*) malloc(ncoef*sizeof(double));

xstart = 1483790.964036;
ystart = 2430868.637727;
xstop = 9.1338e5;
ystop = 9.8015e5;
xdif = xstop - xstart;
ydif = ystop - ystart;
k1 = 0;
while(k1 < sizeofinit)
{
xinit[k1] = xstart + (xdif/(sizeofinit-1))*k1;
yinit[k1] = ystart + (ydif/(sizeofinit-1))*k1;
k1 = k1 + 1;
}
for (k1=0;k1<sizeofinit;k1++)
{
coefficients[k1] = xinit[k1];
coefficients[k1+sizeofinit] = yinit[k1];
}
coefficients[ncoef-1] = 172800; //TMR: for time variable
//linspace(coefficients,0,0,ncoef);
//Done with the download of coefficients
/* Allocate space for breakpoints and initialize */
bps=(double*) malloc(optparam.nbps*sizeof(double));
linspace(bps,0,optparam.HL,optparam.nbps);
/* NTG Memory Variables */
istate=(int*) malloc((ncoef+
optparam.nlic+optparam.nlfc+
optparam.nltc*optparam.nbps+
optparam.nnlic+
optparam.nnltc*optparam.nbps+
optparam.nnlfc)*sizeof(int));
clambda=(double *) malloc((ncoef+
optparam.nlic+optparam.nlfc+
optparam.nltc*optparam.nbps+
optparam.nnlic+optparam.nnltc*optparam.nbps+
optparam.nnlfc)*sizeof(double));
R=(double *) malloc((ncoef+1)*(ncoef+1)*sizeof(double));
/* Set NPSOL options if any */
for(i=0;i<optparam.nnpsol_options;i++)
```

```
npsoloption(optparam.npsol_options[i]);
//line 159 to line 173 is copied from "ntgmultilo.c",try to find the optimal solution
//Call to ntg
//npsoloption("verify level = 3");
npsoloption("Major iteration limit = 3000");
npsoloption("Minor iteration limit = 1500");
npsoloption("Line search tolerance = 0.001");
npsoloption("Feasibility tolerance = 2.e-5");
npsoloption("cold start");
ntg(optparam.nout,bps,optparam.nbps,
optparam.ninterv,knots,ptparam.order,optparam.mult,
optparam.maxderiv,coefficients,optparam.nlic,
optparam.lic_A,optparam.nltc,optparam.ltc_A,
optparam.nlfc,optparam.lfc_A,optparam.nnlic,nlicf,// Function pointer
optparam.nnltc,nltcf,  // Function pointer
optparam.nnlfc,nlfcf,// Function pointer
optparam.ninitialconstrav,optparam.initialconstrav,
optparam.ntrajectoryconstrav,optparam.trajectoryconstrav,
optparam.nfinalconstrav,optparam.finalconstrav,
optparam.lowerb,
optparam.upperb,
optparam.nicf,icf,// Function pointer
optparam.ntcf,tcf,    // Function pointer
optparam.nfcf,fcf,    // Function pointer
optparam.ninitialcostav,optparam.initialcostav,
optparam.ntrajectorycostav,optparam.trajectorycostav,
optparam.nfinalcostav,optparam.finalcostav,
istate,clambda,R,&inform,&objective);
// Get trajectories from B-Spline Coefficients
nTraj=0;
for(i=0;i<optparam.nout;i++)
nTraj += optparam.maxderiv[i];
Traj = (double**) malloc(nPts*sizeof(double));
for(i=0;i<nPts;i++)
Traj[i] = (double*) malloc(nTraj*sizeof(double));

Traj_offset = 0;coef_offset = 0;
Time = (double*) malloc(nPts*sizeof(double));
linspace(Time,0,optparam.HL,nPts);

for(i=0;i<optparam.nout;i++){
for(j=0;j<nPts;j++){
SplineInterp  &Traj[j][Traj_offset],  // Return Variable
Time[j],      // Point at which to evaluate
knots[i],      // Knot sequence
optparam.ninterv[i],        // Number of intervals
&coefficients[coef_offset], NCOEF[i],  // Coefficients
optparam.order[i],optparam.mult[i],
optparam.maxderiv[i]);
}
Traj_offset += optparam.maxderiv[i];
coef_offset += NCOEF[i];
}
Tf = coefficients[ncoef-1];
printf("\n Tf = %f secs = %f minutes  = %f hours\n", Tf, Tf/60, Tf/3600);
// Open File to print data
fp=fopen("TrOldModTest.txt","w");
fprintf(fp," %% time(min)  x(cm)   xd(cm/sec)  y(cm) yd(cm/sec)\n");
for(j=0;j<nPts;j++){
 fprintf(fp,"\n");
}
fcurrent = fopen("CurOldTest.dat","w"); //TMR
float tm;
tm=Time[j]*Tf/3600;
if(tm>24) tm=24;//we only have the data from 1 to 25 hours
for(j=0;j<nPts;j++)
{
GetSplineInfo(Traj[j][0], Traj[j][2],tm, currentdatau, currentdatav);
GetSplineInfo(Traj[j][0], Traj[j][2],tm, currentdatau, currentdatav);
fprintf(fcurrent,"%lf  %lf  %lf  %lf  %lf  %lf %lf %lf \n ",currentdatau[0],
currentdatau[1], currentdatau[2],currentdatau[3], currentdatav[0], currentdatav[1],
currentdatav[2],currentdatav[3]);
}
//To see the values (x,y) upto 3600*14 min = 3600 sec

linspace(Time,0,(double)(50400./Tf),nPts);
Traj_offset = 0;coef_offset = 0;
for(i=0;i<optparam.nout;i++){
for(j=0;j<nPts;j++){
SplineInterp( &Traj[j][Traj_offset],
Time[j],// Point at which to evaluate
knots[i],        // Knot sequence
optparam.ninterv[i], // Number of intervals
&coefficients[coef_offset], NCOEF[i],  // Coefficients
optparam.order[i],
optparam.mult[i],
optparam.maxderiv[i]);
}
Traj_offset += optparam.maxderiv[i];
coef_offset += NCOEF[i];
}
printf("\n At t=25(from10) hours x, y = %f %f 0.0\n\n",
Traj[nPts-1][0]-1000, Traj[nPts-1][2]-1000);
printf("\n At t=25(from10) hours x, y = %f %f 172800\n\n",
Traj[nPts-1][0], Traj[nPts-1][2]);
// Open File to print data
fp2 = fopen("opt1upto25.dat","w");
if(fp2 ==NULL){
fprintf(stderr,"Can't open file opt1upto60min.dat for writing\n");
exit(-1);}
// Print to File
for(j=0;j<nPts;j++){
fprintf(fp2,"%lf ",Time[j]*Tf/60);
fprintf(fp2,"%lf  %lf  %lf  %lf",Traj[j][0],
Traj[j][1]/Tf, Traj[j][2], Traj[j][3]/Tf);//x,xd,y,yd
fprintf(fp2,"\n");
}
fcurrent2 = fopen("currentinfoupto60min.dat","w");
if(fcurrent2 == NULL){
fprintf(stderr,"Can't open file opt1upto60min.dat for writing\n");
exit(-1);
}
for(j=0;j<nPts;j++){
GetSplineInfo(Traj[j][0], Traj[j][2],Time[j]*Tf/3600, currentdatau,
currentdatav);
GetSplineInfo(Traj[j][0], Traj[j][2],Time[j]*Tf/3600, currentdatau,
currentdatav);
fprintf(fcurrent2,"%lf  %lf  %lf  %lf  %lf  %lf \n ",currentdatau[0],
currentdatau[1], currentdatau[2],currentdatau[3], currentdatav[0],
currentdatav[1],currentdatav[2],currentdatav[3]);
}
fclose(fp);
fclose(fp2);
fclose(fcurrent);
fclose(fcurrent2);
free(Time);
free(NCOEF);
for(i=0;i<optparam.nbps;i++)
free(Traj[i]);
free(Traj);
free(istate);
free(clambda);
free(R);
free(bps);
for (i=0;i<optparam.nout;i++)
free(knots[i]);
free(coefficients);
return 0;
}

//opt1.sub.h
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#ifndef _opt1_autocode_header_
#define _opt1_autocode_header_
#define z1 zp[0][0]
#define z1d zp[0][1]
#define z2 zp[1][0]
#define z2d zp[1][1]
#define z3 zp[2][0]
#define PI              3.14159
#define NBPS            100

int check = 1;
int GetBcoef(float B[], float knots[], int k, int lengthknots, float data)
 {
//This function calculates the Basis functions of the B-spline
//function at given data point and get the relative location of input data
// data is one dimension, x or y or t
//INPUTS:
//  knots: knot points (found by MATLAB program sigfitspline2.m)
//  k    : order of the B-spline basis function
//OUTPUT:
//  index: location of the data respect to knot sequence
//  B[] : Basis function values at a given data for  for degree k
//METHOD:
//Multidimensional tensor B-spline products can be calculated explicitly by using:
//  z=f(x,y,t)=sum(i=1,n1){sum(j=1,n2){sum(k=1,n3){{B_(i,k1)(x)*B_(j,k2)(y)B_(k,k3)(t)*Aijk}}}
//  In general :
//  The definition of B-Spline basis function is
//  B_(i,k)(u) = {(u -knots_u(i))/(knots_u(i+k-1) - knots_u(i))}*B_(i,k-1)(u) +
//  {(knots_u(i+k) - u)/(knots_u(i+k) - knots_u(i+1))}*B_(i+1,k-1)(u)
//  B_(i,1)(u) = 1 if  knots_u(i)  <=  u  <=  knots_u(i+1)
//  0 otherwise
//  Note: In B_(i,k)(u), if denominator of any term is equal to zero, that term is SET to zero!
// For example if knots_u(i+k) = knots_u(i+1) OR knots_u(i+k-1) = knots_u(i)
```

```
//
// Please check the matlab file omasplinefit3d.m
// B-spline coefficient matrix A and knots were found by using the MATLAB program omasplinefit3d.m
// For details please see, "A Practical Guide to Splines", revised edition, Carl de Boor, 2001, page 111
int d1, j, r; //dummy variables
int index = 0; //in our case x=elavation, y=azimuth
float tempsaved;
float tempterm ;
//Add 1 to start the index from 1 as in Carl De Boor's notation
float deltaR[4+1];       //Right Side  need constant,can not wait for k transfered from the outside
float deltaL[4+1];          //Left Side
for (j=0; j<=k; j++) {B[j] = 0; deltaR[j] = 0; deltaL[j] = 0;}
for (d1=1; d1<lengthknots; d1++){
if ((data >= knots[d1]) && (data <knots[d1+1]) )
index = d1;//show this commond has been executed
}
B[1] = 1; //Skip B[0] to agree with Carl De Boor's notation
for (j=1; j<k; j++)
{
deltaR[j] = knots[index+j] - data;
deltaL[j] = data - knots[index+1-j];
tempsaved = 0;
for (r=1; r<=j; r++)
{
tempterm = B[r]/(deltaR[r] + deltaL[j+1-r]);
B[r] = tempsaved + deltaR[r]*tempterm;
tempsaved = deltaL[j+1-r]*tempterm;
}
B[j+1] = tempsaved;
}
return index;
}


void EvaluateSpline(float result[], float A[][22][27], float Bx[],
float By[],float Bt[], int xindex, int yindex,int tindex, int kx, int ky,int kt)
{
//This function evaluates the tensor product spline function value at given data point
//INPUTS:
//A[][][]: Coefficients of the B-spline fit, found by the MATLAB program, omasplinefit3d.m
// Bx[] : Basis function values at a given data for  x for degree kx
// By[] : Basis function values at a given data for  y for degree ky
// Bt[] : Basis function values at a given data for  t for degree kt
// xindex: location of the x-data respect to knot sequence
// yindex: location of the y-data respect to knot sequence
// tindex: location of the t-data respect to knot sequence
// kx    : order of the B-spline basis function for x
// ky    : order of the B-spline basis function for y
// kt    : order of the B-spline basis function for t
//OUTPUT:
//  result[]: Evaluation of the tensor product spline function is given in result[0]
//METHOD:
// Multidimensional tensor B-spline products can be calculated explicitly by using:
//  f(x,y,t)=sum(i=1,n1){sum(j=1,n2){sum(k=1,n3){B_(i,k1)(x)*B_(j,k2)(y)*B_(k,k3)(t)*Aijk}}}
//   n1,n2,n3: # of coefficients of x,y,t direction respectively
//  k1,k2,k3: degree of splines of x,y,t direction respectively
//
// Specifically,
//   For knots sequences: knots_x(r1) <= x < knots_x(r1+1)
//       knots_y(r2) <= y < knots_y(r2+1)
```

```
//
// f(x,y,t) = sum(i=r1-k1+1,r1){sum(j=r2-k2+1,r2){sum(k=r3-k3+1,r3)
      {B_(i,k1)(x)*B_(j,k2)(y)*B_(k,k3)(t)*Aijk}}
//
// For details please see, "A Practical Guide to Splines", revised edition, Carl de Boor,
 2001, page 117
// values at the begining of knots sequences, and also for the B[], and
// dB[] matrices first elemnts are not used
// which corresponds to indices = 0.
// Please check the matlab file bformcheck3.m and omasplinefit3d.m programs
// B-spline coefficient matrix A and knots were found by using the MATLAB program
 omasplinefit3d.m

int d1, d2,d3; //dummy variables
result[0] = 0; //Signature value
for (d3=tindex-kt+1;d3<=tindex;d3++)
{
for (d1=xindex-kx+1; d1<=xindex; d1++)
{
for (d2=yindex-ky+1; d2<=yindex; d2++)
{
result[0] = result[0] + Bx[d1-xindex+kx]
*By[d2-yindex+ky]*Bt[d3-tindex+kt]*A[d1-1][d2-1][d3-1];
}
}
}

void EvaluateSplineDerivatives(float result[], float A[][22][27],
float Bx[], float By[],float Bt[], float dBx[], float dBy[],float
dBt[], int xindex, int yindex,int tindex, int kx, int ky,int kt,
float knotsx[], float knotsy[],float knotst[])
{
//TMR: This function evaluates the first derivatives of tensor product
// spline functions at given data point
//INPUTS:
//  A[][][]: Coefficients of the B-spline fit, found by the MATLAB
program, omasplinefit3d.m
//  Bx[] : Basis function values at a given data for  x for degree kx
//  By[] : Basis function values at a given data for  y for degree ky
//     Bt[] : Basis function values at a given data for  y for degree ky
//  dBx[] : Basis function values at a given data for  x for degree kx-1
//  dBy[] : Basis function values at a given data for  y for degree ky-1
//     dBt[] : Basis function values at a given data for  t for degree kt-1
//  xindex: location of the x-data respect to knot sequence
//  yindex: location of the y-data respect to knot sequence
//     tindex: location of the t-data respect to knot sequence
//  kx     : order of the B-spline basis function for x
//  ky     : order of the B-spline basis function for y
//     kt     : order of the B-spline basis function for t
// knotsx: knot points for x-direction (found by MATLAB program omasplinefit3d.m)
//knotsy: knot points for y-direction (found by MATLAB program omasplinefit3d.m)
//     knotst: knot points for t-direction (found by MATLAB program omasplinefit3d.m)
//OUTPUT:
//  result[]: Evaluation of the first derivatives of tensor product spline
function are given in result[1] & [2]&[3]
//METHOD:
//  Multidimensional tensor B-spline 1st derivatives can be calculated
//explicitly by using:
// f(x,y,t)=sum(i=1,n1){sum(j=1,n2){sum(k=1,n3){B_(i,k1)(x)*B_(j,k2)(y)*B_(k,k3)*Aijk}}}
// n1,n2,n3: # of coefficients in x,y,t direction
// k1,k2,k3: degree of splines in x,y,t direction
//
//Specifically,
//For knots sequences: knots_x(r1) <= x < knots_x(r1+1)
//       knots_y(r2) <= y < knots_y(r2+1)
//                        knots_t(r3) <= t < knots_t(r3+1)
//
// f(x,y,t) = sum(i=r1-k1+1,r1){sum(j=r2-k2+1,r2){sum(k=r3-k3+1,r3)*{B_(i,k1)(x)
//*B_(j,k2)(y)*B_(k,k3)(t)*Aijk}}}
//
// Dx(f(x,y,t)) = sum(i=r1-k1+2,r1){sum(j=r2-k2+1,r2)
//{sum(k=r3-k3+1,r3)*{(k1-1)*(A(i,j,k) - //A(i-1,j,k))
*B_(i,k1-1)(x)*B_(j,k2)(y)*B_(k,k3)(t)}}}
//     ------------------
//(knotsx(i+k1-1) - knotsx(i))
//Dy(f(x,y,t)) = sum(i=r1-k1+1,r1){sum(j=r2-k2+2,r2)
//{sum(k=r3-k3+1,r3)*{(k2-1)*(A(i,j,k) - A(i,j-1,k))
//*B_(i,k1)(x)*B_(j,k2-1)(y)*B_(k,k3)(t)}}}
//  ------------------
//(knotsy(j+k2-1) - knotsy(j))
//
//Dt(f(x,y,t)) = sum(i=r1-k1+1,r1){sum(j=r2-k2+1,r2)
//{sum(k=r3-k3+2,r3)*{(k3-1)*(A(i,j,k) - A(i,j,k-1))
//*B_(i,k1)(x)*B_(j,k2)(y)*B_(k,k3-1)(t)}}}
// ------------------
//(knotsy(k+k3-1) - knotsy(k))
//
// Please check the matlab file bformcheck3.m and sigfitspline2.m programs
// B-spline coefficient matrix A and knots were found by using
//the MATLAB program sigfitspline2.m or for the probability
// pdfitspline3.m
int d1, d2,d3; //dummy variables
result[1] = 0; //Derivative value in x-direction
result[2] = 0; //Derivative value in y-direction
result[3] = 0;      //Derivative value in t-direction added on Feb 7 2007
for (d1=xindex-kx+2; d1<=xindex; d1++)
{
for (d2=yindex-ky+1; d2<=yindex; d2++)
{
for(d3=tindex-kt+1;d3<=tindex; d3++)
{
result[1] = result[1] + (kx-1)/(knotsx[d1+kx-1] - knotsx[d1])
*(A[d1-1][d2-1][d3-1] - A[d1-2][d2-1][d3-1])*dBx[d1-xindex+kx-1]*
By[d2-yindex+ky]*Bt[d3-tindex+kt];
}
}
}
for (d1=xindex-kx+1; d1<=xindex; d1++)
{
for (d2=yindex-ky+2; d2<=yindex; d2++)
{
for(d3=tindex-kt+1;d3<=tindex; d3++)
{
result[2] = result[2]+(ky-1)/(knotsy[d2+ky-1]-knotsy[d2])
*(A[d1-1][d2-1][d3-1]-A[d1-1][d2-2][d3-1])
*Bx[d1-xindex+kx]*dBy[d2-yindex+ky-1]*Bt[d3-tindex+kt];
}                             }
}
```

```
for (d1=xindex-kx+1; d1<=xindex; d1++)
{
for (d2=yindex-ky+2; d2<=yindex; d2++)
{
for(d3=tindex-kt+2;d3<=tindex; d3++)
result[3] = result[3]+(kt-1)/(knotst[d3+kt-1]
-knotst[d3])*(A[d1-1][d2-1][d3-1]-A[d1-1][d2-1][d3-2])
*Bx[d1-xindex+kx]*By[d2-yindex+ky]*dBt[d3-tindex+kt-1];
}
}
}

void GetSplineInfo(float xdataf, float ydataf,float tdataf,
float currentinfou[4], float currentinfov[4])
{
//TMR: This function gives the current value, and its first
//derivatives respect to xdataf and ydataf.
//INPUTS:
//  xdataf: xdata data value
//  ydataf : ydata data value
//  tdataf : tdata data value
//OUTPUT:
//currentinfo[0] : current value
//currentinfo[1] : derivative of the current respect to xdataf angle given in radians.
//currentinfo[2] : derivative of the current respect to ydata angle given in radians.
//currentinfo[3] : derivative of the current respect to tdata angle given in hour
//This part has to changed for each different spline fit --->
FILE *fileAu, *fileknotsux, *fileknotsuy, *fileknotsut;
FILE *fileAv, *fileknotsvx, *fileknotsvy, *fileknotsvt;
int k1,k2,k3,j;
//Add 1 to start the index from 1 as in Carl De Boor’s notation
float dummy = 1e20;
const int lengthknotsx = 36+1 ;
const int lengthknotsy = 26+1;
const int lengthknotst = 31+1   ;  //knotsut=31, t=1~25,expanded to t after
//we use t=1,2,3 for test ,knots of t are 1 1 1 1 2 3 3 3 3
int nx = 32; //# of coefficients in the tensor product B-spline expression,
//n=[n1 n2,n3]
int ny = 22;
int nt = 27;
const int kx = 4;   //degree of B-spline polynomials, k=[k1 k2,k3]
const int ky = 4;
const int kt = 4;
//For current value u(x,y,t)
static  float knotsux[37];//36+1
static  float knotsuy[27];//26+1
static  float knotsut[32];//31+1
static  float Au[32][22][27];
//For current value v(x,y,t)
static  float knotsvx[37];
static  float knotsvy[27];
static  float knotsvt[32];
static  float Av[32][22][27];
//Variables for the evaluation of the tensor product B-splines
//Add 1 to start the index from 1 as in Carl De Boor’s notation
float Bx[kx+1];//For [Bi-k+1,k(el) ... Bi,k(el)] : total k
//elements active for each ti <= x <ti+1
float By[ky+1];//Values of basis functions, Bi, will be in B[1],
//..., B[k+1], skip the B[0] value!
```

```
float Bt[kt+1];
float xindex ;
float yindex ;
float tindex ;
//Variables for the evaluation of the 1st derivative of tensor product B-splines
int dkx = kx - 1;
int dky = ky - 1;
int dkt = kt - 1;
float dBx[kx];    //original float dBx[dkx+1] the error
float dBy[ky];
float dBt[kt];
//Read knots and coefs matrix here
if (check == 1){
knotsux[0] = dummy;
knotsuy[0] = dummy;
knotsut[0] = dummy;
fileknotsux = fopen("KnotsInfo/knotsux","r");
for (k1 = 0; k1 < lengthknotsx-1; k1++)
fscanf(fileknotsux, "%f ", &knotsux[k1+1]);
fclose(fileknotsux);
fileknotsuy = fopen("KnotsInfo/knotsuy","r");
for (k1 = 0; k1 < lengthknotsy-1; k1++)
fscanf(fileknotsuy, "%f ", &knotsuy[k1+1]);
fclose(fileknotsuy);
fileknotsut = fopen("KnotsInfo/knotsut","r");
for (k1 = 0; k1< lengthknotst-1; k1++)
fscanf(fileknotsut,"%f ", &knotsut[k1+1]);
fclose(fileknotsut);
fileAu = fopen("KnotsInfo/Au.txt","r");
for (k3 = 0; k3 < nt; k3++)
for (k1 = 0; k1 < nx; k1++)
for (k2 =0; k2 < ny; k2++)
fscanf(fileAu, "%f,", &Au[k1][k2][k3]);
fclose(fileAu);

knotsvx[0] = dummy;
knotsvy[0] = dummy;
knotsvt[0] = dummy;
fileknotsvx = fopen("KnotsInfo/knotsvx","r");
for (k1 = 0; k1 < lengthknotsx-1; k1++)
fscanf(fileknotsvx, "%f ", &knotsvx[k1+1]);
fclose(fileknotsvx);
fileknotsvy = fopen("KnotsInfo/knotsvy","r");
for (k1 = 0; k1 < lengthknotsy-1; k1++)
fscanf(fileknotsvy, "%f ", &knotsvy[k1+1]);
fclose(fileknotsvy);
fileknotsvt = fopen("KnotsInfo/knotsvt","r");
for (k1 = 0; k1 < lengthknotst-1; k1++)
fscanf(fileknotsvt, "%f ", &knotsvt[k1+1]);
fclose(fileknotsvt);
int   scan_result;
fileAv = fopen("KnotsInfo/Av.txt","r");
for (k3 = 0; k3 < nt; k3++)
for (k1 = 0; k1 < nx; k1++)
for (k2 = 0; k2 < ny; k2++){
scan_result=fscanf(fileAv, "%f,", &Av[k1][k2][k3]);
if (scan_result==-1)
puts("scan fail");
}
```

```
fclose(fileAv);
}
//Done with reading
//Evaluate the tensor product B-splines
for (j=0; j<=kx; j++) {Bx[j] = 0;}
for (j=0; j<=ky; j++) {By[j] = 0;}
for (j=0; j<=kt; j++) {Bt[j] = 0;}
xindex = GetBcoef(Bx, knotsux, kx, lengthknotsx, xdataf);
yindex = GetBcoef(By, knotsuy, ky, lengthknotsy, ydataf);
tindex = GetBcoef(Bt, knotsut, kt, lengthknotst, tdataf);
EvaluateSpline(currentinfou, Au, Bx, By,Bt, xindex, yindex,
tindex, kx, ky,kt);
xindex = GetBcoef(Bx, knotsvx, kx, lengthknotsx, xdataf);
yindex = GetBcoef(By, knotsvy, ky, lengthknotsy, ydataf);
tindex = GetBcoef(Bt, knotsvt, kt, lengthknotst, tdataf);
EvaluateSpline(currentinfov, Av, Bx, By,Bt, xindex, yindex,
tindex, kx,ky,kt);
//Evaluate the 1st derivatives of tensor product B-splines
for (j=0; j<=dkx; j++) {dBx[j] = 0;}
for (j=0; j<=dky; j++) {dBy[j] = 0;}
for (j=0; j<=dkt; j++) {dBt[j] = 0;}
xindex = GetBcoef(dBx, knotsux, dkx, lengthknotsx, xdataf);
yindex = GetBcoef(dBy, knotsuy, dky, lengthknotsy, ydataf);
tindex = GetBcoef(dBt, knotsut, dkt, lengthknotst, tdataf);
EvaluateSplineDerivatives(currentinfou, Au, Bx, By,Bt, dBx,
dBy,dBt, xindex, yindex,tindex, kx, ky,kt, knotsux, knotsuy,knotsut);
xindex = GetBcoef(dBx, knotsvx, dkx, lengthknotsx, xdataf);
yindex = GetBcoef(dBy, knotsvy, dky, lengthknotsy, ydataf);
tindex = GetBcoef(dBt, knotsut, dkt, lengthknotst, tdataf);
EvaluateSplineDerivatives(currentinfov, Av, Bx, By,Bt, dBx,
dBy,dBt, xindex, yindex,tindex, kx, ky,kt, knotsvx, knotsvy,knotsvt);
}
/* Nonlinear Initial Constraint */
/* ============================ */
void nlicf(int *mode, int *nstate, double *f, double **df, double **zp){
}
/* Nonlinear Trajectory Constraint */
/* =============================== */
void nltcf(int *mode, int *nstate, int *i, double *f, double **df,
   double **zp)
float currentinfou[4] = {0};
float currentinfov[4] = {0};
float tempz1 = 0;
float tempz2 = 0;
int tp1,tp2;
float time,tao;
//TMR: Set up internal time parameters   //copy from ntgmultilo to test the internal time
tp1  = *i;
tp2  = NBPS;
tao = tp1*1.0/tp2;
time=tao*z3/3600;//hours
if(time>24)time=24;
//  printf("\n the internal time in the nltcf is= %f \n",time);
int k1;
if(*mode==0 || *mode==2){
//TMR: Variables for GetSplineInfo() function (B-spline fit):
for (k1=0; k1 < 4; k1++)
{
currentinfou[k1] = 0; //TMR: Need to set to zero before each call
```

```
currentinfov[k1] = 0;
}
tempz1 = (float)(z1);
tempz2 = (float)(z2);
GetSplineInfo(tempz1, tempz2,time, currentinfou, currentinfov);
GetSplineInfo(tempz1, tempz2,time, currentinfou, currentinfov);
check++;
//printf("\n z1 = %f, z2 = %f,  yobstacle[0] = %f,  z2-yobstacle[0] = %f,
yobstacle[1] = %f", z1, z2, yobstacle[0], z2-yobstacle[0], yobstacle[1]);
f[0] = (z1d/z3 - currentinfou[0])*(z1d/z3 - currentinfou[0]) + (z2d/z3 -
currentinfov[0])*(z2d/z3 - currentinfov[0]);
}
if(*mode==1 || *mode==2){
df[0][0] = -2*(z1d/z3 - currentinfou[0])*currentinfou[1] -2*(z2d/z3 -
currentinfov[0])*currentinfov[1];  /* wrt z1 */
df[0][1] =  2*(z1d/z3 - currentinfou[0])*(1/z3);  /* wrt z1d */
df[0][2] = -2*(z1d/z3 - currentinfou[0])*currentinfou[2]
-2*(z2d/z3 - currentinfov[0])*currentinfov[2];  /* wrt z2 */
df[0][3] =  2*(z2d/z3 - currentinfov[0])*(1/z3);  /* wrt z2d */
//df[0][4] =-2*(z1d/z3-currentinfou[0])*(z1d/z3/z3)
-2*(z2d/z3-currentinfov[0])*(z2d/z3/z3);
df[0][4] =0;//regard T constant
//df[0][4] = -2*(z1d/z3 - currentinfou[0])*(z1d/z3/z3+
tao/3600*currentinfou[3]) -2*(z2d/z3 - currentinfov[0])
*(z2d/z3/z3+tao/3600*currentinfov[3]);//assume du/dT is not equal to 0
//df[0][4] = -2*(z1d/z3 - currentinfou[0])*(z1d/z3/z3  -
2*(z2d/z3 - currentinfov[0])*(z2d/z3/z3);
//assume du/dT=0,T is constant ,t=tao*T
}
}
/* Nonlinear Final Constraint */
/* ========================== */
void nlfcf(int *mode, int *nstate, double *f, double **df, double **zp){
}
/* Initial Cost */
/* ============ */
void icf(int *mode, int *nstate, double *f, double *df, double **zp){
double  Wq=0.1;
if(*mode==0 || *mode==2){
*f = Wq*z3;
}
if(*mode==1 || *mode==2){
df[0] = 0;
df[1] = 0;
df[2] = 0;
df[3] = 0;
df[4] = Wq;
    }
}
/* Trajectory Cost */
/* =============== */
void tcf(int *mode, int *nstate, int *i, double *f, double *df, double **zp){
float currentinfou[4] = {0};
float currentinfov[4] = {0};
float tempz1 = 0;
float tempz2 = 0;
int k1;
double Wu =0;
int tp1,tp2;
```

```
float time,tao;
tp1  = *i;
tp2  = NBPS;
tao= tp1*1.0/tp2;
time=tao*z3/3600;//hours
if(time>24)time=24;
// printf("\n the internal time in the tcf is= %f \n",time);
if(*mode==0 || *mode==2){
for (k1=0; k1 < 4; k1++)
{
currentinfou[k1] = 0;
currentinfov[k1] = 0;
}
tempz1 = (float)(z1);
tempz2 = (float)(z2);
GetSplineInfo(tempz1, tempz2,time, currentinfou, currentinfov);
GetSplineInfo(tempz1, tempz2,time, currentinfou, currentinfov);
check++;
f[0] = Wu*z3*(z1d/z3 - currentinfou[0])*(z1d/z3 -
currentinfou[0]) + Wu*z3*(z2d/z3 - currentinfov[0])*(z2d/z3 - currentinfov[0]);
printf("f[0]/Wu=%lf\n",f[0]/Wu);
}
if(*mode==1 || *mode==2){
df[0] = -2*Wu*z3*(z1d/z3 - currentinfou[0])*currentinfou[1]
-2*Wu*z3*(z2d/z3 - currentinfov[0])*currentinfov[1];  /* wrt z1 */
df[1] =  2*Wu*(z1d/z3 - currentinfou[0]);  /* wrt z1d */
df[2] =-2*Wu*z3*(z1d/z3 - currentinfou[0])*currentinfou[2]
-2*Wu*z3*(z2d/z3 - currentinfov[0])*currentinfov[2];  /* wrt z2 */
df[3] =  2*Wu*(z2d/z3 - currentinfov[0]);  /* wrt z2d */
//df[4] =-2*Wu*((z1d/z3 - currentinfou[0])*(z1d/z3)
+(z2d/z3 - currentinfov[0])*(z2d/z3))+Wu*((z1d/z3
- currentinfou[0])*(z1d/z3 - currentinfou[0]) + (z2d/z3
- currentinfov[0])*(z2d/z3 - currentinfov[0]));
df[4]=0;// Regard T is constant here
//df[4] = -2*Wu*z3*(z1d/z3 - currentinfou[0])*(z1d/z3/z3
+tao/3600*currentinfou[3]) -2*Wu*(z2d/z3 - currentinfov[0])
*(z2d/z3/z3+tao/3600*currentinfov[3]) + Wu*((z1d/z3 -
currentinfou[0])*(z1d/z3 - currentinfou[0]) + (z2d/z3
- currentinfov[0])*(z2d/z3 - currentinfov[0]));//assume du/dT=tao*du/dt
//df[4] = -2*Wu*z3*(z1d/z3 - currentinfou[0])*z1d/z3/z3
-2*Wu*(z2d/z3 - currentinfov[0])*z2d/z3/z3 + Wu*((z1d/z3
 - currentinfou[0])*(z1d/z3 - currentinfou[0]) + (z2d/z3
 - currentinfov[0])*(z2d/z3 - currentinfov[0])); //du/dT=0 even t=tao*T
}
}
/* Final Cost */
/* ========== */
void fcf(int *mode, int *nstate, double *f, double *df, double **zp){
}
#endif

//opt1.inp
% Trajectory Definitions
% =====================
    NOUT 3
    NINTERV 30 30 1
    MULT  3 3 1
    ORDER  6 6 1
    MAXDERIV  2 2 1
```

```
% Horizon Length
% ==============
    HL  1
% Number of break points
% =====================
    NBPS 100
% Define Linear Initial Constraints
% ================================
%//Thee are for min time trajectories after for zone 24 and on
%//6/25/04
    NLIC 3
    LIC_LB
    1620806.642201 3223724.674343   0
    LIC_UB
    1620906.642201 3223824.674343   172800
    LIC_A
%   x   xd  y  yd  T
% =================
    1   0   0   0  0
    0   0   1   0  0
    0   0   0   0  1
% Define Linear Trajectory Constraints
% ================================
%    NLTC 2
%
%    LTC_LB
%    1 9.3619e5
%
%    LTC_UB
%   4.7254e6 4.2757e6
%
%    LTC_A
%   x   xd  y  yd  T
% =================
%    1   0   0   0  0
%    0   0   1   0  0
% Define Linear Final Constraints
% ================================
    NLFC 2
    LFC_LB
    9.1338e5      9.8015e5
    LFC_UB
    9.1348e5      9.8025e5
    LFC_A
%   x   xd  y  yd  T
% =================
    1   0   0   0  0
    0   0   1   0  0
% Define Nonlinear Trajectory Constraints
% =======================================
    NNLTC 1
    NLTC_LB
    0
    NLTC_UB
    1600
    NTRAJECTORYCONSTRAV 5
    TRAJECTORYCONSTRAV  0 0 0 1 1 0 1 1 2 0
% Define Initial Cost
% ====================
    NICF 1
    NINITIALCOSTAV 1
    INITIALCOSTAV  2 0
% Define Trajectory Cost
% ======================
    NTCF 1
    NTRAJECTORYCOSTAV 5
    TRAJECTORYCOSTAV  0 0 0 1 1 0 1 1 2 0
% Define NPSOL Options
% ====================
    NNPSOLOPTION 2
    NPSOLOPTION    NOLIST
    NPSOLOPTION    Print Level 5
  % NPSOLOPTION    Major Iteration Limit 100
  % NPSOLOPTION    Minor Iteration Limit 100
//opt1.makefile
 #Make file is modified by zwz according to the Makefile
 of vanderpol Jan #25 2007
#$NTGDIR  and $NTGMLDIR has to be defined
NTGDIR = ..
NTGINCDIR = $(NTGDIR)/include
NTGLIBDIR = $(NTGDIR)/lib
CC=gcc -g -I $(NTGINCDIR)
CFLAGS= -O3 -m32 -I $(NTGINCDIR)  -L $(NTGLIBDIR) -include opt1.sub.h
LIB= -lm -lgfortran  -lntg -lnpsol -lpgs -lg2c
opt1: opt1.main.o
$(CC) $(CFLAGS) -L $(NTGLIBDIR) -o opt1 opt1.main.c $(LIB)
```

APPENDIX III

NTG program for a JPL Aerobot

```
//This program is modified to generate the optimal trajectory for
// an aerobot which is travelling from (0,0,0) to (200,200,200)
// The wind profile is assumed to have the layer format which the wind speed
// is changing between layers while it remains the same in the same layer
// for example, in layer one, z is from (0,50), u=10,v=0 (x and y direction
// wind speeds), in layer two, z is from (50,100), u=10,v=10. In layer three
// u=8, v=0, in layer four, u=5,v=-10.

// The aerobot has three inputs, the model of aerobot is simplified and modifed
//from the underwater glider model. The three inputs are V (forward velocity),
//dtheta/dt (the change of orietation), W (ascend or descent velocity,upward
//and downward). The cost function is WtT+\int(0,T) K1V+K2dtheta/dt+K3W,K1,K2,
//K3 are three coefficients for the three control inputs.The constraints are
//the start point (0,0,0), the destination point (200,200,200). The trajectory
//constraints are the velocity bounds(forward and upward) and orientation
//change bounds (one).Here the trajectory is 3D trajectory whihc has the information
//of x(t), y(t), z(t), assuming around 10 m/s for the aerobot, initial Tf=50 s
//opt1.main.c for the aerobot
#include <stdlib.h>
#include <math.h>/* math functions */
#include <ntg.h>/* main NTG declarations */
#include <time.h>             /*get the time*/
#include <ParseInputFile.c>
int main(int argc, char *argv[])
{
OPTPARAM optparam;
int i,j,sum,k1;
char *fname;
FILE *fp;
FILE *fder;
double **Traj,*Time;
int nTraj, Traj_offset,coef_offset, nPts=30;
float windU[9] = {0}; //TMR
float windV[9] = {0}; //TM
double Tf = 50;
FILE *fxinit, *fyinit;
const int sizeofinit = 93; //test interval 50 ,153
float xinit[93]={0};
float yinit[93]={0};
float zinit[93]={0};  //3D trajectory
float tauinit[93]={0}; //control input
//float winit[93]={0};

//static float xinit[sizeofinit]={0,0};

float xstart, xstop, xdif, ystart, ystop, ydif,zstart,zstop,zdif;
double **knots;/* knot points, list of times for each output */
```

```
int nbps;
double *bps;
double **lic,**lfc, **ltc;
/* initial guess size = sum over each output ninterv*(order-mult)+mult */
int ncoef;
int *NCOEF;
double *coefficients;
int *istate;
double *clambda;
double *R;
int inform;
double objective;

if (argc!=2){
printf("\n\tUsage: %s inputfile.inp\n\n",argv[0]);
exit(-1);
}
knots=(double **) malloc(optparam.nout*sizeof(double*));

for (i=0;i<optparam.nout;i++){
knots[i]=(double *) malloc((optparam.ninterv[i]+1)*sizeof(double));
linspace(knots[i], 0, optparam.HL, optparam.ninterv[i]+1);
}
ncoef = 0;
NCOEF = (int*) malloc(optparam.nout*sizeof(int));
for(i=0;i<optparam.nout;i++){
ncoef = ncoef + (optparam.ninterv[i]*(optparam.order[i]
-optparam.mult[i])+optparam.mult[i]);
NCOEF[i] = optparam.ninterv[i]*(optparam.order[i]
-optparam.mult[i])+optparam.mult[i];
}
/*  Initial guess for coefficients (all 0s)*/
coefficients=(double*) malloc(ncoef*sizeof(double));
xstart=0;
ystart=0;
zstart=0;
xstop=200;
ystop=200;
zstop=200;
xdif = xstop - xstart;
ydif = ystop - ystart;
zdif = zstop - zstart;
k1 = 0;
while(k1 < sizeofinit)
{
xinit[k1] = xstart + (xdif/(sizeofinit-1))*k1;
yinit[k1] = ystart + (ydif/(sizeofinit-1))*k1;
zinit[k1] = zstart + (zdif/(sizeofinit-1))*k1;
tauinit[k1]=-1;
k1 = k1 + 1;

}

//Change initial guess, try to make the optimal solution
//to be similiar with the one from DMOC, the initial guess
//is separated into 4 segments
while(k1<13){
xinit[k1]=10-(2.0/12)*k1;
yinit[k1]=0-0.8/12*k1;
k1=k1+1;
}
int kk2;
kk2=0;
while(kk2<20){
xinit[k1]=8-2.0/19*kk2;
yinit[k1]=-0.8+1.8/19*kk2;
kk2=kk2+1;
k1=k1+1;
}
int kk3;
kk3=0;
while(kk3<30)
{
xinit[k1]=6+2.0/29*kk3;
yinit[k1]=1+3.0/29*kk3;
kk3=kk3+1;
k1=k1+1;
}
int kk4;
kk4=0;
while(kk4<30)
{
xinit[k1]=8+7.0/29*kk4;
yinit[k1]=4-2.0/29*kk4;
kk4=kk4+1;
k1=k1+1;
}*/
for (k1=0;k1<sizeofinit;k1++){
coefficients[k1] = xinit[k1];
coefficients[k1+sizeofinit] = yinit[k1];
coefficients[k1+sizeofinit*2] = zinit[k1];
coefficients[k1+sizeofinit*3] = tauinit[k1];
// coefficients[k1+sizeofinit*3] = winit[k1];
}
coefficients[ncoef-1]=100;
//linspace(coefficients,0,0,ncoef);
//Done with the download of coefficients
/* Allocate space for breakpoints and initialize */
bps=(double*) malloc(optparam.nbps*sizeof(double));
linspace(bps,0,optparam.HL,optparam.nbps);
/* NTG Memory Variables */
istate=(int*) malloc((ncoef+optparam.nlic
+optparam.nlfc+optparam.nltc*optparam.nbps+
optparam.nnlic+optparam.nnltc*optparam.nbps+
optparam.nnlfc)*sizeof(int));
clambda=(double *) malloc((ncoef+
optparam.nlic+optparam.nlfc+
optparam.nltc*optparam.nbps+
optparam.nnlic+optparam.nnltc*optparam.nbps+
optparam.nnlfc)*sizeof(double));
R=(double *) malloc((ncoef+1)*(ncoef+1)*sizeof(double));

/* Set NPSOL options if any */for(i=0;i<optparam.nnpsol_options;i++)
npsoloption(optparam.npsol_options[i]);
//line 159 to line 173 is copied from "ntgmultilo.c",try to find the optimal solution
//Call to ntg
npsoloption("Major iteration limit = 3000");
npsoloption("Minor iteration limit = 1500");
npsoloption("Line search tolerance = 0.001");
npsoloption("Feasibility tolerance = 2.e-5");
npsoloption("istate=1");
npsoloption("Line search tolerance =0.001");
npsoloption("Hessian =Yes");
ntg(optparam.nout,
bps,optparam.nbps,optparam.ninterv,knots,
optparam.order,optparam.mult,optparam.maxderiv,
coefficients,
optparam.nlic,optparam.lic_A,
optparam.nltc,optparam.ltc_A,
optparam.nlfc,optparam.lfc_A,
optparam.nnlic,nlicf,// Function pointer
optparam.nnltc,nltcf,  // Function pointe
optparam.nnlfc,nlfcf,// Function pointer
optparam.ninitialconstrav,optparam.initialconstrav,
optparam.ntrajectoryconstrav,optparam.trajectoryconstrav,
optparam.nfinalconstrav,optparam.finalconstrav,
optparam.lowerb,optparam.upperb,
optparam.nicf,icf,// Function pointer
optparam.ntcf,tcf,     // Function pointer
optparam.nfcf,fcf,     // Function pointer
optparam.ninitialcostav,optparam.initialcostav,
optparam.ntrajectorycostav,optparam.trajectorycostav,
optparam.nfinalcostav,optparam.finalcostav,
istate,clambda,R,&inform,&objective);
//PrintVector("coef1",coefficients,ncoef);
// Get trajectories from B-Spline Coefficients
nTraj=0;
for(i=0;i<optparam.nout;i++) nTraj += optparam.maxderiv[i];
Traj = (double**) malloc(nPts*sizeof(double));
for(i=0;i<nPts;i++)Traj[i] = (double*) malloc(nTraj*sizeof(double));
Traj_offset = 0;coef_offset = 0;
Time = (double*) malloc(nPts*sizeof(double));
linspace(Time,0,optparam.HL,nPts);
for(i=0;i<optparam.nout;i++){
for(j=0;j<nPts;j++){
SplineInterp( &Traj[j][Traj_offset],  // Return Variable
Time[j],      // Point at which to evaluate
knots[i],      // Knot sequence
optparam.ninterv[i],        // Number of intervals
&coefficients[coef_offset], NCOEF[i],  // Coefficients
optparam.order[i],
optparam.mult[i],
optparam.maxderiv[i]);

}

Traj_offset += optparam.maxderiv[i];
coef_offset += NCOEF[i];
}

Tf = coefficients[ncoef-1];  //TMR: Tf in seconds
printf("\n Tf = %f secs = %f minutes  = %f hours\n", Tf, Tf/60, Tf/3600);
fp=fopen("TrAeroEL.txt","w");
fprintf(fp," %% time(min)  x(cm) xd(cm/sec)
xdd(cm/sec/sec)  y(cm) yd(cm/sec) ydd(cm/sec/sec)
z zd zdd windU  windV Tf orient tau\n"); // Print to File
float tm;
```

```
for(j=0;j<nPts;j++){
GetWindInfo(Traj[j][6], windU, windV); //z...Traj[j][6]
fprintf(fp,"%lf %lf  %lf  %lf  %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf
%lf\n",Time[j]*Tf,Traj[j][0], Traj[j][1]/Tf, Traj[j][2]/Tf/Tf,
Traj[j][3],Traj[j][4]/Tf,Traj[j][5]/Tf/Tf, Traj[j][6], Traj[j][7]/Tf,
Traj[j][8]/Tf/Tf, windU[0],windV[0], Tf,
atan2(Traj[j][4]/Tf-windV[0],Traj[j][1]/Tf-windU[0]), Traj[j][9]);
}
fclose(fp);
free(Time);
free(NCOEF);
free(Traj);
free(istate);
free(clambda);
free(R);
free(bps);
for (i=0;i<optparam.nout;i++)
free(knots[i]);
free(coefficients);
return 0;

}
```

```
//opt1.sub.h for the aerobot
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#ifndef _opt1_autocode_header_
#define _opt1_autocode_header_
#define z1 zp[0][0]
#define z1d zp[0][1]
#define z1dd    zp[0][2]   //xdd
#define z2 zp[1][0]
#define z2d zp[1][1]
#define z2dd    zp[1][2]  //ydd
#define z3      zp[2][0]  //z
#define z3d     zp[2][1]
#define z3dd    zp[2][2]
#define z4      zp[3][0]  //tauc  control variable
#define z4d     zp[3][1]
#define z4dd    zp[3][2]
#define z5 zp[4][0]  //T
#define PI              3.14159
int check = 1;
/* Function to define the wind velocities */
void GetWindInfo(float zdataf, float windU[9], float windV[9]){
if (zdataf>=0 && zdataf<50){
windU[0]=10;
windV[0]=10;}
else if (zdataf>=50 && zdataf<100){
windU[0]=-10;
windV[0]=10;}
else if (zdataf>=100 && zdataf<150){
windU[0]=10;
windV[0]=-10;}
else if (zdataf>=150 && zdataf<=300){
windU[0]=0;
windV[0]=-10;}
```

```
else
printf("out of bound, the limit is 0-300 in zdataf");
}
/* Nonlinear Initial Constraint */
/* ============================ */
void nlicf(int *mode, int *nstate, double *f, double **df, double **zp){
}
/* Nonlinear Trajectory Constraint */
/* =============================== */
void nltcf(int *mode, int *nstate, int *i, double *f,
 double **df, double **zp){
float windU[9] = {0};  // u,ux,uy,uz,ut,utx,uty,utz,utt
float windV[9] = {0};
float tempz = 0;
float Cx=1; float Cy=1; float Cz=1; // Drag and lift coefficients
int k1;
if(*mode==0 || *mode==2){
for (k1=0; k1 < 9; k1++)
{
windU[k1] = 0; //TMR: Need to set to zero before each call
windV[k1] = 0;
}
tempz = (float)(z3);
GetWindInfo(tempz, windU, windV);
check++;
f[0] = (z1d/z5 - windU[0])*(z1d/z5 - windU[0]) + (z2d/z5
 - windV[0])*(z2d/z5 - windV[0])+z3d*z3d/z5/z5; //V, forward velocity
// f[1] = (z2dd/z5/z5-windV[3])/(z1d/z5-windU[0]);//dtheta/dt
//Euler-Lagrange equationsnow I donot know how I got these equations
f[1]=-z1dd/z5/z5 + 0.5*10.049*pow(z1d/z5,2)*Cx+z4*z1dd/pow(z5,2);
//z4 is the control variable
f[2]=-z2dd/z5/z5 + 0.5*10.049*pow(z2d/z5,2)*Cy+z4*z2dd/pow(z5,2);
f[3]=-9.8-z3dd/z5/z5 + 0.5*10.049*pow(z3d/z5,2)*Cz + z4*z3dd/pow(z5,2);
}
if(*mode==1 || *mode==2){
df[0][0]=-0;
df[0][1] = 2*(z1d/z5- windU[0])*(1/z5);  /* wrt z1d */
df[0][2] = 0; /*wrt z1dd*/
df[0][3] = 0;
df[0][4] = 2*(z2d/z5- windV[0])*(1/z5); /* wrt z2d */
df[0][5] = 0; /*wrt z2dd*/
df[0][6] = 0;  //for z3
df[0][7] = 2*z3d/z5/z5;
df[0][8] = 0;
df[0][9] = 0;  // for z4
df[0][10]= 0;
df[0][11]= 0;
df[0][12]= 0;
df[1][0]=0;
df[1][1]=2*0.5*10.049*Cx*z1d/z5/z5;
df[1][2]=z4/z5/z5; //wrt z1dd
df[1][3]=0;
df[1][4]=0;
df[1][5]=0;
df[1][6]=0;//z3
df[1][7]=0;
df[1][8]=0;
df[1][9] = z1dd/pow(z5,2);  //z4
df[1][10]= 0;  //z4d
df[1][11]= 0;  //z4dd
df[1][12]=0;
df[2][0]=0;
df[2][1]=0;
df[2][2] = 0; //wrt z1dd
df[2][3]=0;
df[2][4] =2*0.5*10.049*z2d/z5/z5;
df[2][5]=z4/pow(z5,2);
df[2][6] = 0;//z3
df[2][7] = 0;
df[2][8] = 0;
df[2][9] =z2dd/pow(z5,2);  //z4
df[2][10] =0;  //z4d
df[2][11] =0;  //z4dd
df[2][12]=0;
df[3][0]=0;
df[3][1]=0;
df[3][2] = 0; //wrt z1dd
df[3][3]=0;
df[3][4] = 0;
df[3][5]=0;
df[3][6] = 0;//z3
df[3][7] = 2*0.5*10.049*z3d/z5/z5;
df[3][8] = -1/z5/z5 + z4/pow(z5,2);
df[3][9] = z3dd/pow(z5,2); //z5,T
df[3][10]=0;
df[3][11]=0;
df[3][12]=0;
}
}

/* Nonlinear Final Constraint */
/* ========================= */
void nlfcf(int *mode, int *nstate, double *f, double **df, double **zp){
}
/* Initial Cost */
/* ============ */
void icf(int *mode, int *nstate, double *f, double *df, double **zp){
double  Wq=1;
if(*mode==0 || *mode==2){
*f = Wq*z5;
}
if(*mode==1 || *mode==2){
df[0] = 0;
df[1] = 0;
df[2] = 0;
df[3] = 0;
df[4] = 0;
df[5] = 0;
df[6] = 0;
df[7] = 0;
df[8] = 0;
df[9] =0;  //T is variable for intial cost funtion,
//so df/dT is not equal to 0
df[10] =0;  //z4
df[11] =0;  //z4
df[12] =Wq; //z5
}
}
```

```
/* Trajectory Cost */
/* =============== */
void tcf(int *mode, int *nstate, int *i, double *f,
double *df, double **zp){
float windU[9] = {0};
float windV[9] = {0};
int k1;
float tempz;
int tp1,tp2,NBPS;
NBPS=100;
float time;
tp1  = *i;
tp2  = NBPS;
float tao,Wu;
tao= tp1*1.0/tp2;
time =tao*z5/3600;
Wu=0;  //100000 for min E
if(*mode==0 || *mode==2){
for (k1=0; k1 < 9; k1++){
windU[k1] = 0; //TMR: Need to set to zero before each call
windV[k1] = 0;
}
tempz = (float)(z3);
GetWindInfo(tempz,  windU, windV);
check++;
// printf("\n tempz =%f, u=%f,v=%f \n",tempz, windU[0],windV[1]);
f[0]=Wu*z5*pow((z1d/z5-windU[0]),2)+
Wu*z5*pow((z2d/z5-windV[0]),2)+
Wu*z5*pow(z3d/z5,2)+Wu*z4*z4*(z1dd*z1dd+z2dd*z2dd+z3dd*z3dd)/pow(z5,3);
f[0]/wu=%lf\n",pow((z1d/z5-windU[0]),2)+pow((z2d/z5-windV[0]),2)
+(z1dd*z1dd+z2dd*z2dd)/pow(z5,3)+z3d*z3d/z5/z5);
}
if(*mode==1 || *mode==2){
df[0]=0;
df[1] =2*Wu*(z1d/z5-windU[0]);  // wrt z1d
df[2]=2*Wu*z4*z4*z1dd/pow(z5,3);
df[3]=0;
df[4] = 2*Wu*(z2d/z5-windV[0]);  //wrt z2d
df[5] = 2*Wu*z4*z4*z2dd/pow(z5,3);
df[6] = 0; // z3
df[7] = 2*Wu*z3d/z5;
df[8] = 2*Wu*z4*z4*z3dd/pow(z5,3);;
df[9] =2*Wu*z4*z5*(z1dd*z1dd+
z2dd*z2dd+z3dd*z3dd)/pow(z5,3);    //z4
df[10] =0;  //z4d
df[11] =0;  //z4dd
df[12]=0;
}
}

/* Final Cost */
/* ========== */
void fcf(int *mode, int *nstate, double *f, double *df, double **zp){
}
#endif

//opt1.inp for the aerobot
% Trajectory Definitions
% =======================
    NOUT 5
    NINTERV 30 30 30  30  1  % x,y,z, control, t
    MULT  3 3 3  3 1  % this should mean "smoothiness"
    ORDER  6 6 6  6 1
    MAXDERIV  3 3 3 3  1
% Horizon Length
% =============
    HL  1
% Number of break points
% =====================
    NBPS 100
% Define Linear Initial Constraints
% =================================
   NLIC 5  %number of initial constraints
   LIC_LB
   0  0  0  -100000 0  %x,y, z, tau, t
   LIC_UB
   0   0  0  100000 100
   LIC_A
% x   xd  xdd  y  yd  ydd z zd zdd  tau td tdd T
% =================
  1   0   0    0   0   0  0  0  0    0  0   0  0
  0   0   0    1   0   0  0  0  0    0  0   0  0
  0   0   0    0   0   0  1  0  0    0  0   0  0
  0   0   0    0   0   0  0  0  0    1  0   0  0
  0   0   0    0   0   0  0  0  0    0  0   0  1
    % initial velocity is -10 (x direction)
% Define Linear Final Constraints
% ===============================
    NLFC 3
    LFC_LB
    200    200   200    %final destination
    LFC_UB
    200    200   200       %
    LFC_A
%   x   xd  xdd  y  yd  ydd z zd zdd tau td tdd   T
% ==========================
  1   0   0    0   0   0    0  0  0 0 0  0   0
  0   0   0    1   0   0    0  0  0 0 0  0   0
  0   0   0    0   0   0    1  0  0 0 0  0   0
% Define Linear Trajectory Constraints
%=====================================
   NLTC 3
   LTC_LB
   0      0      0
   LTC_UB
   300    300   300
   LTC_A
%   x   xd  xdd  y  yd  ydd  z  zd  zdd  tau td tdd  T % coef
% ===========================
   1    0    0  0  0   0   0   0    0   0    0  0  0
   0    0    0  1  0   0   0   0    0   0    0  0  0
   0    0    0  0  0   0   1   0    0   0    0  0  0
% Define Nonlinear Trajectory Constraints
% =====================================
    NNLTC  4
    NLTC_LB
    0  0  0 0
    NLTC_UB
    400 0 0 0
    NTRAJECTORYCONSTRAV 8
    TRAJECTORYCONSTRAV    0 1   0 2   1 1  1 2   2 1  2 2   3 0   4 0
% Define Initial Cost
% =====================
    NICF 1
    NINITIALCOSTAV 1
    INITIALCOSTAV  4 0
% Define Trajectory Cost
% =====================
    NTCF 1
    NTRAJECTORYCOSTAV 8
    TRAJECTORYCOSTAV  0 1   0 2   1 1   1 2   2 1   2 2   3 0   4 0
% Define NPSOL Options
% ===================
    NNPSOLOPTION 2
    NPSOLOPTION    NOLIST
    NPSOLOPTION    Print Level 5
```

APPENDIX IV

DMOC program for a glider in a B-spline ocean model

```
#DMOC program for the glider is written in AMPL, the solver both can be IPOPT
#or NPSOL, in order to compare the NTG with DMOC, we choose NPSOL since NTG
#is using NPSOL as its solver. In this program DMOC needs call the B-spline
# ocean current model thus B·spline function should act as a user-defined
# function. In Unutnu, the makefile for adding user-defined function is also
#attached here while B-spline function program is not completely listed here
#considering it is just modified from the foregoing appedix.

#test.mod, funcadd.h. funcadd.c makefile.linux are needed
#first, make -f makefile.linux to create amplfunc.dll
#then ./ampl test.mod to get the solution

option solver npsol;
#option npsol_options "iterations=3000";
#option ipopt_options "halt_on_ampl_error yes";
#option ipopt_options 'max_iter=100';
#option ipopt_options "max_iter=3000";
#option ipopt_options "constr_viol_tol=1e-5";
#option ipopt_options "tol=1e-5";

function usplineinfo;
function vsplineinfo;

param N:=51; # number of knots in the trajectory

set POS_NODES := {0..N-1};
set VEL_NODES := {0..N-2};
# a = (a_x, a_y) and b = (b_x, b_y) are positions of start and final points
param a_x := 1620806.642201; #cm,(xstart-xref)*scalefactor from opt1.inp
param a_y := 3223724.674343;
param b_x := 913380.0;
param b_y := 980150.0;
param T0 := 172800.0;  #Initial trial final time
param  m :=1.0;
param kno :=N;
param h0 :=T0/(kno-1);

var x {i in POS_NODES};
var y {i in POS_NODES};
var tau {i in POS_NODES}>=-1000000, <=1000000;     # Control in every knot
var lambda >=0.99, <=1.01;
var h=h0;  #final time is fixed to T=48 hour
var  T=T0; #final time is fixed to T=48 hour
var q1p {i in VEL_NODES} = (x[i+1]-x[i])/h;
var q2p {i in VEL_NODES} = (y[i+1]-y[i])/h;
var q1m {i in VEL_NODES} = 0.5*(x[i]+x[i+1]);
var q2m {i in VEL_NODES} = 0.5*(y[i]+y[i+1]);
var u {i in VEL_NODES};
var v {i in VEL_NODES};
var wx {i in VEL_NODES} =usplineinfo(x[i],y[i],h*i/3600.0,u[i],v[i]);
var wy {i in VEL_NODES}= vsplineinfo(x[i],y[i],h*i/3600.0,u[i],v[i]);
var taum {i in VEL_NODES} = tau[i];
#derivative of kinetic energy w.r.t \dot{q}
var KEq1p {i in VEL_NODES} = m*(q1p[i]-wx[i]);
var KEq2p {i in VEL_NODES} = m*(q2p[i]-wy[i]);
#derivative of kinetic energy w.r.t q
param KEq1 {i in VEL_NODES} := 0;
param KEq2 {i in VEL_NODES} := 0;
#potential
param Vq1 {i in VEL_NODES} :=0;
param Vq2 {i in VEL_NODES} :=0;
#discrete forces
var force1_plus {i in VEL_NODES} = -taum[i]*(q2p[i]-wy[i]);
var force1_minus {i in VEL_NODES} = -taum[i]*(q2p[i]-wy[i]);
var force2_plus {i in VEL_NODES} = taum[i]*(q1p[i]-wx[i]);
var force2_minus {i in VEL_NODES} = taum[i]*(q1p[i]-wx[i]);
#minimize the control energy and time
param Wu:=1;
minimize force_energy:
sum{j in 0..N-2} 0.5*(force1_plus[j]*force1_plus[j]+
force1_minus[j]*force1_minus[j]+force2_plus[j]*force2_plus[j]+
force2_minus[j]*force2_minus[j])*h*Wu; #+0.5*T;

#Subject to constraints, there are 104 equality constraints due to EL equations
#Here, the position is considered differently, so there are 100 EL equations needed

#Starting and final point
subject to x_left_anchor: x[0] = a_x;
subject to y_left_anchor: y[0] = a_y;
subject to x_right_anchor: x[N-1] = b_x;
subject to y_right_anchor: y[N-1] = b_y;

#Starting velocity (momentum)
# No constraints on the final velocity
subject to Euler_Lagrange_x {j in 0..N-3}:
 -KEq1p[j+1] + KEq1p[j] + 0.5*h*(KEq1[j+1]+KEq1[j]) + 0.5*h*(Vq1[j+1]+Vq1[j])
  + 0.5*h*force1_plus[j] + 0.5*h*force1_minus[j+1] = 0;
subject to Euler_Lagrange_y {j in 0..N-3}:
-KEq2p[j+1] + KEq2p[j] + 0.5*h*(KEq2[j+1]+KEq2[j]) + 0.5*h*(Vq2[j+1]+Vq2[j])
  + 0.5*h*force2_plus[j] + 0.5*h*force2_minus[j+1] = 0;
param xref:=-122.32458;
param yref:=36.5658;
param scale:=11126067;

#Start point  guess 1
#Let initial guess to be the trajecotry on the left side of the straight line
let {j in 0..19} x[j] :=(j/19)*(-122.3-xref)*scale + (1-j/19)*(-122.178-xref)*scale;
let {j in 0..19} y[j] :=(j/19)*(36.75-yref)*scale+(1-j/19)*(36.8557-yref)*scale;
let {j in 20..N-1} x[j] := (j-20)/(N-1-20)*(-122.242-xref)*scale
 + (1-(j-20)/(N-1-20))*(-122.3-xref)*scale;
let {j in 20..N-1} y[j] := (j-20)/(N-1-20)*(36.6535-yref)*scale
+ (1-(j-20)/(N-1-20))*(36.75-yref)*scale;
let {j in 0..N-1} tau[j] := -1;
let lambda:= 1;

display x,y >DMOCinitL.txt;
solve;
```

```
display x, y, q1p, q2p, tau>trajL.txt;
display q1p,q2p,tau,wx,wy;
display T/3600;
display force_energy;

//funcadd.c

/* sample funcadd */
#include "math.h" /* for sqrt */
#include "funcadd.h" /* includes "stdio1.h" */

int check1 = 0;  // for usplineinfo
int check2 = 0; //for vsplineinfo
void EvaluateSpline(float result[], float A[][22][27], float Bx[],
float By[],float Bt[], int xindex,
int yindex,int tindex, int kx, int ky,int kt) ;
void EvaluateSplineDerivatives(float result[], float A[][22][27],
 float Bx[], float By[],float Bt[], float dBx[], float dBy[],float
dBt[], int xindex, int yindex,int tindex, int kx, int ky,int kt,
 float knotsx[], float knotsy[],float knotst[]);
int GetBcoef(float B[], float knots[], int k, int lengthknots, float data);
real u;
real v;
static real
usplineinfo(register arglist *al)
{
real xdataf,ydataf,tdataf;
        float currentinfou[7] = {0};
        float currentinfov[7] = {0};
        real x, z;
real *d, *de, *ra;
int *at, i,  n;
char *se;
const char *sym;
AmplExports *ae = al->AE; /* for fprintf and strtod */
if ((n = al->n) <= 0)
return 0;
at = al->at;
ra = al->ra;
d = de = al->derivs;
x = 0.;

        check1++;
//make the knots info just updated once
//modied by weizhong zhang, to make the complex b-spline ocean current
//model work
xdataf=ra[0];  //input x as the first variable
ydataf=ra[1];
tdataf=ra[2];
//checky=ra[at[5]];

if(tdataf>24) tdataf=24; //we only have 24 time zone data,
// when t>24, assume the current stays constant at t=24.
....
}
 void
funcadd(AmplExports *ae){
/* Insert calls on addfunc here... */
/* Arg 3, called argtype, can be 0 or 1:
 * 0 ==> force all arguments to be numeric
 * 1 ==> pass both symbolic and numeric arguments.
 *
 * Arg 4, called nargs, is interpretted as follows:
 * >=  0 ==> the function has exactly nargs arguments
 * <= -1 ==> the function has >= -(nargs+1) arguments.
 *
 * Arg 5, funcinfo, is passed to the functions in struct arglist;
 * it is not used in these examples, so we just pass 0.
 */
addfunc("usplineinfo",(rfunc)usplineinfo, 0,5,0);
        addfunc("vsplineinfo",(rfunc)vsplineinfo, 0,5,0);
}
}
```

```
#Makefile for adding function to create amplfunc.dll
#which is from AMPL, thank David M. Gay's help to make it work
# For Linux
.SUFFIXES: .c .o
# $S = ampl/solvers directory
S = ..
CC = cc
CFLAGS = -I$S -O2
.c.o:
$(CC) -c $(CFLAGS) $*.c
amplfunc.dll: funcadd.c
$(CC) -c $(CFLAGS) -fPIC funcadd.c
$(CC) -shared -o amplfunc.dll funcadd.o
## Sample solver creation...
# $(myobjects) = list of .o files
myobjects = ....
mysolver: $(myobjects)
$(CC) -o mysolver $(myobjects) $S/amplsolver.a -lm -ldl
```

APPENDIX V

DMOC program for a JPL Aerobot

```
#DMOC program for generating trajectories of a JPL Aerobot
#in the defined wind profile with consideration of aerodynamics.
#This problem constraints are Euler-Lagrange equations which are
#from the perspective of energy rather than Newton's perspective.

#test.mod, funcadd.h. funcadd.c makefile.linux are needed
#first, make -f makefile.linux to create amplfunc.dll
#then ./ampl test.mod to get the solution.

#Program test.mod for the Aerobot

option solver npsol;
#option npsol_options 'iterations = 3000';
#option npsol_options 'Minor iteration limit = 1500';
#option npsol_options 'linesearch= 1.0e-8';
#option npsol_options 'Linear Feasibility tolerance = 1.0e-8';
#option npsol_options 'Nonlinear Feasibility tolerance = 1.0e-8';
#option npsol_options 'cold start';
#option npsol_options 'Optimality tolerance =1.0e-8';

# Find optimal trajectories for a JPL Aerobot
# From (0,0,0) to (200,200,200) in the wind

function uWind;
function vWind;
param N:=51; # number of knots in the trajectory
set POS_NODES := {0..N-1};
set VEL_NODES := {0..N-2};
set ACE_NODES := {0..N-3};
# a = (a_x, a_y, a_z) and b = (b_x, b_y,b_z) are positions of
# start and final points
param a_x := 0;
param a_y := 0;
param a_z := 0;
param b_x := 200;
param b_y := 200;
param b_z := 200;
param T0 := 100;  #Initial trial final time
param  m :=1.0;
param kno :=N;
param h0 :=T0/(kno-1);
#Initial speed
param xinip:= 0;
param yinip:= 0;
param zinip:= 0;
#Bounds on variables
var x {i in POS_NODES}>=0, <=300;
var y {i in POS_NODES}>=0, <=300;
var z {i in POS_NODES}>=0, <=300;
var tau {i in POS_NODES}>=-100000, <=100000;     # Control in every knot
var lambda >=0, <=1;          #
#var h=h0*lambda;
#var T=T0*lambda;
var h=h0;
var T=T0;
var q1p {i in VEL_NODES} = (x[i+1]-x[i])/h;
var q2p {i in VEL_NODES} = (y[i+1]-y[i])/h;
var q3p {i in VEL_NODES} = (z[i+1]-z[i])/h;
# Acceleration
var q1pp {i in ACE_NODES} = (q1p[i+1]-q1p[i])/h;
var q2pp {i in ACE_NODES} = (q2p[i+1]-q2p[i])/h;
var q3pp {i in ACE_NODES} = (q3p[i+1]-q3p[i])/h;
var q1m {i in VEL_NODES} = 0.5*(x[i]+x[i+1]);
var q2m {i in VEL_NODES} = 0.5*(y[i]+y[i+1]);
var q3m {i in VEL_NODES} = 0.5*(z[i]+z[i+1]);
var wx  {i in VEL_NODES} = uWind(x[i],y[i],z[i]);
var wy  {i in VEL_NODES} = vWind(x[i],y[i],z[i]);
# no wind velocity in z direction
var taum {i in VEL_NODES} = tau[i];
#minimize the control energy and time
param Wu:=1;
minimize force_energy: sum{j in 0..N-3} ((q2p[j]-wy[j])
*(q2p[j]-wy[j])+(q1p[j]-wx[j])*(q1p[j]
-wx[j])+q3p[j]*q3p[j]+(q1pp[j]*q1pp[j]+q2pp[j]*q2pp[j]
+q3pp[j]*q3pp[j])*tau[j]*tau[j])*h*Wu;
#Subject to constraints, there are 104 equality constraints
#due to EL equations
# Here, the position is considered differently, so there
#are 100 EL equations needed

#Starting and final point
subject to x_left_anchor: x[0] = a_x;
subject to y_left_anchor: y[0] = a_y;
subject to z_left_anchor: z[0] = a_z;
subject to x_right_anchor: x[N-1] = b_x;
subject to y_right_anchor: y[N-1] = b_y;
subject to z_right_anchor: z[N-1] = b_z;
#The constraints in the trajectory, there are 96(48 knots) EL equations
#are needed to be satisfied considering it is controlled by the gyroscopic force

subject to Velocity_total {j in 0..N-2}:
(q1p[j]-wx[j])*(q1p[j]-wx[j])+(q2p[j]-wy[j])*(q2p[j]-wy[j])
 + q3p[j]*q3p[j]<=400;

subject to Euler_Lagrange_x {j in 0..N-3}:
-q1pp[j] + tau[j]*q1pp[j] + 0.5*10.049*q1p[j]*q1p[j] = 0;

subject to Euler_Lagrange_y {j in 0..N-3}:
-q2pp[j] + tau[j]*q2pp[j] + 0.5*10.049*q2p[j]*q2p[j] = 0;

subject to Euler_Lagrange_z {j in 0..N-3}:
-9.8-q3pp[j] + tau[j]*q3pp[j] + 0.5*10.049*q3p[j]*q3p[j] = 0;

#Start point
let {j in 0..N-1} x[j] := (j/N)*b_x + (1-j/N)*a_x;
let {j in 0..N-1} y[j] := (j/N)*b_y + (1-j/N)*a_y;
let {j in 0..N-1} z[j] := (j/N)*b_z + (1-j/N)*a_z;
let {j in 0..N-1} tau[j] := -1;
let lambda:= 1;

display x,y,z >initdmoc.txt;
solve;
display x,y,z,tau >traj.txt;
display x,y,z,wx,wy,q1p,q2p,q3p,tau,T,force_energy;

/*funcadd */
#include "math.h" /* for sqrt */
#include "funcadd.h" /* includes "stdio1.h" */
static real
uWind(arglist *al) /* sqrt(x*x + y*y) */
{
real xdataf,ydataf,zdataf;
float windU[7] = {0};
float windV[7] = {0};
real x, z;
real *d, *de, *ra;
int *at, i,  n;
char *se;
const char *sym;
AmplExports *ae = al->AE; /* for fprintf and strtod */
if ((n = al->n) <= 0) return 0;
at = al->at;
ra = al->ra;
d = de = al->derivs;
x = 0.;
xdataf=ra[0];  //input x as the first variable
ydataf=ra[1];
zdataf=ra[2];
if (zdataf>=0 && zdataf<50){
windU[0]=10;
windV[0]=10;}
else if (zdataf>=50 && zdataf<100){
windU[0]=-10;
windV[0]=10;}
else if (zdataf>=100 && zdataf<150){
windU[0]=10;
windV[0]=-10;}
else if (zdataf>=150 && zdataf<=300){
windU[0]=0;
windV[0]=-10;}
else
printf("out of bound, the limit is 0-300 in zdataf");
return windU[0];
}
static real vWind(arglist *al) {
real xdataf,ydataf,zdataf;
float windU[7] = {0};
float windV[7] = {0};
real x, z;
real *d, *de, *ra;
int *at, i,  n;
char *se;
const char *sym;
AmplExports *ae = al->AE; /* for fprintf and strtod */
if ((n = al->n) <= 0) return 0;
at = al->at;
ra = al->ra;
d = de = al->derivs;
x = 0.;
xdataf=ra[0];  //input x as the first variable
ydataf=ra[1];
zdataf=ra[2];
if (zdataf>=0 && zdataf<50){
windU[0]=10;
windV[0]=10;}
else if (zdataf>=50 && zdataf<100){
windU[0]=-10;
windV[0]=10;}
else if (zdataf>=100 && zdataf<150){
windU[0]=10;
windV[0]=-10;}
else if (zdataf>=150 && zdataf<=300){
windU[0]=0;
windV[0]=-10;}
else
printf("out of bound, the limit is 0-300 in zdataf");
return windV[0];
}
void funcadd(AmplExports *ae){
/* Insert calls on addfunc here... */
/* Arg 3, called argtype, can be 0 or 1:
 * 0 ==> force all arguments to be numeric
 * 1 ==> pass both symbolic and numeric arguments.
 *
 * Arg 4, called nargs, is interpretted as follows:
 * >=  0 ==> the function has exactly nargs arguments
 * <= -1 ==> the function has >= -(nargs+1) arguments.
 *
 * Arg 5, funcinfo, is passed to the functions in struct arglist;
 * it is not used in these examples, so we just pass 0.
 */
addfunc("uWind",(ufunc*)uWind, 1, -1,0);
        addfunc("vWind",(ufunc*)vWind, 1, -1,0);
}
```

APPENDIX VI

MATLAB program for generating ODE45 trajectories

```
% By Weizhong Zhang under directions of Dr Jerry E. Marsden
% To use MATLAB ODE45 (Ordinary Differential Equation Solver) to
% reintegrate the trajectory

% Derive the equations of motion for an underwater glider controlled
% by gyroscopic forces, put optimal control input from DMOC or NTG
% into the equations of motion as fixed controlinput, then reintegrate
% the trajectory using ode45 to check the DMOC and NTG solutions

% Cost function
% \int_0^t{\tau^2(\dot{y}-v)^2 + \tau^2(\dot{x}-u)^2}  (5)

% Equations of motion
% \ddot{x}=-\tau*(\dot{y}-v) + \dot{u}  (1)
% \ddot{y}= \tau*(\dot{x}-u) + \dot{v}  (2)

% Initial Conditions
% x(0)=10, y(0)=0, \dot{x}(0)=-10, \dot{y}(0)=-10
% Final Conditions
% x(1)=15, y(1)=2
% Current  u=0.1x, v=0


% On the DMOC and NTG trajectories, there are 51 points, it means there are
% 51 tau values, for every tau value, there is a corresponding x value,
% thus y value. Thus, we can get a new sets of x, y values according to the
% tau value

% For ODE45 to solve the problem, the differential equation should be a
% first order equation, it needs to define a function glider_motion_equ()
clc;
clear all;

i=1;  % for tau(i)
p01=[10 -10 0 -10];
options = odeset(’RelTol’,1e-6,’AbsTol’,1e-6);
[t1,p1] = ode45(@glider_motion_equ_ntg,[0:1/2000:i/50],p01,options);
save (’ODEntg/p1.txt’,’p1’,’-ascii’);
save (’ODEntg/t1.txt’,’t1’,’-ascii’);

% T=1/51 for tau(2), the initial condition should be changed to the final
% condition from last calculation
for i=2:50
var=strcat(’p’,num2str(i-1));
name=strcat(’/home/weizhong/Desktop/MatlabWork/ODEntg/’,var,’.txt’);
% Initial condition for the next ODE solution, t=i/50, i=1:50
p_mid=load(name);
[m,n]=size(p_mid);
```

```
p_init=p_mid(m,:); % the last row from the last ODE solution

% Begin to generate next file
tname=strcat('t',num2str(i),'.txt');
pname=strcat('p',num2str(i),'.txt');
ftname=strcat('ODEntg/',tname);
fpname=strcat('ODEntg/',pname);
[tname,pname]=ode45(@glider_motion_equ_ntg,[(i-1)/50:1/2000:i/50],p_init,options);
save(fpname,'pname','-ascii');
save(ftname,'tname','-ascii');
end


function dp = glider_motion_equ_ntg(t,p)
% This function defines the equations of motion for an underwater glider
% p is a vector defined as [x, dx/dt, y, dy/dt]
%Equations of motion
% \ddot{x}=-\tau*(\dot{y}-v) + \dot{u}  (1)
% \ddot{y}= \tau*(\dot{x}-u) + \dot{v}  (2)
% dp/dt=[0        1    0    0;
%        0       -0.1 0 -\tau;
%        0        0    0    1;
%       -0.1\tau \tau 0    0;]p
TrNTGSimCur=zeros(51,9);
TrNTGSimCur=load('/home/weizhong/Desktop/NTG/TraSimCur/TrSimCur_st.txt');
for j=1:50
if t>=(j-1)/50 && t<=j/50
    i=j;
end
end
tau(i)=TrNTGSimCur(i,9);
dp = zeros(size(p));
dp(1)=p(2);
dp(2)=-0.1*p(2)-tau(i)*p(4);
dp(3)=p(4);
dp(4)=-0.1*tau(i)*p(1)+tau(i)*p(2);

% Plot trajectory from ODE 45 when the control input as the optimal
% solution from DMOC
% p=[x,dx/dt,y.dy/dt]
clear all;
for i=1:50
var=strcat('p',num2str(i));
dname=strcat('/home/weizhong/Desktop/MatlabWork/ODEdmoc/',var,'.txt');
gname=strcat('/home/weizhong/Desktop/MatlabWork/ODEntg/',var,'.txt');
pd=load(dname);
gd=load(gname);
hold on;
plot(pd(:,1),pd(:,3),'-r','LineWidth',2);
hold on;
plot(gd(:,1),gd(:,3),'-','LineWidth',2);
end
TrDMOCSimCur=zeros(51,6);
TrNTGSimCur=zeros(51,9);
DMOCini=zeros(51,3);
TrDMOCSimCur=load('/home/weizhong/Desktop/DMOCnpsol/TestSimCurrent/traj_st.txt');
DMOCinit=load('/home/weizhong/Desktop/DMOCnpsol/TestSimCurrent/xinitdmoc_st.txt');
TrNTGSimCur=load('/home/weizhong/Desktop/NTG/TraSimCur/TrSimCur_st.txt');
hold on;plot(TrDMOCSimCur(:,2),TrDMOCSimCur(:,3),'g','LineWidth',3);
hold on;plot(TrNTGSimCur(:,2),TrNTGSimCur(:,5),'b','LineWidth',3);
hold on;plot(DMOCinit(:,2),DMOCinit(:,3),'--','LineWidth',2);
h=legend('DMOC ODE45 Solution','NTG ODE45 Solution','DMOC Traj','NTG Traj','Init Guess',5);
set(h,'Interpreter','none');
hold on;plot(10,0,'o');
hold on;plot(15,2,'*');
title('Trajectory from DMOC, NTG versus their ODE 45 solutions','FontSize',30);
set(gca,'FontSize',30); xlabel('X','FontSize',30);ylabel('Y','FontSize',30);
grid on;
```

APPENDIX VII

Program to obtain real-time coordiates of a draganflyer

```
// In order to get the real-time coordinates of a draganflyer,
// first, you need to install Real Time SDK, your project must
// include the VrtSDK10ex.h header file and link with VrtSDK10ex.lib.
// To link with VrtSDK10ex.lib, you need to "add existing item" in "Project"
// zwz.cpp : main project file.
#include "stdafx.h"
#include <iostream>
#include "VrtSDK10ex.h"
#include <string.h>
//using namespace System;
using namespace std;
using std::string;
char *chIpAddress="192.168.1.230";
char *chError="Error Message";
int main()   //array<System::String ^> ^args
{
if(ViconConnect(chIpAddress)==true){
cout <<"RTE is connected.\n";
}
else
cout<<"RTE is not connected\n";
if(ViconIsConnected()==true){
cout<<"ViconIsConnected\n";
}
// Get one frame of data
bool param=false;
ViconGetFrame(param);  //param is set to be false, this is for future features
if(ViconGetFrame(param)==true){
cout<<"Get one frame of data\n";
}
else
cout<<"Cannot get data of frame";
int BodyCount=0;
int *p_BodyCount=&BodyCount;
//Get Number bodies
ViconGetNumBodies(p_BodyCount);
cout<<"The number of bodies is:"<<BodyCount<<"\n";
// Get Marker Number
int MarkerCount;
int *nMarkerCount=&MarkerCount;
if(ViconGetNumMarkers(nMarkerCount)==true){
cout<<"the number of markers is:"<<MarkerCount<<"\n";
}
// Get Marker Name
int nMarkerNum[5]={0,1,2,3,4}; //0 based number of the marker
string chName[5]; //contain five marker names
char *p_chName=&chName[0];
cout<<"old p_chName "<<*p_chName<<endl;
for (i=0;i++;i<5){
if(ViconGetMarkerName(nMarkerNum[i],p_chName++)==true){
cout<<"the marker number is:"<<nMarkerNum[i]<<"\n";
cout<<"the marker name is:"<<*p_chName<<"\n";}
else{
cout<<"cannot retrieve marker names"<<"\n";
cout<<"the marker name is:"<<*p_chName<<"\n";}
}
cout<<"new p_chName "<<*p_chName<<endl;
char *a_pName=&chName; //get marker name, why there is only one marker
float aX=0; float aY=0;
float aZ=0; long  aV=0;
float *a_rX=&aX; float *a_rY=&aY;
float *a_rZ=&aZ; long  *a_rV=&aV;
int nBodyNum=0; //zero based
char chBName[10]={’c’,’c’};
char *p_chBName=&chBName;
if(ViconGetBodyName(nBodyNum,p_chBName)==true){
cout<<"The body number is:"<<nBodyNum<<"\n";
cout<<"The body name is:"<<chBName<<"\n";
}
else{cout<<"cannot retrieve body names"<<"\n";}
char *p_BName=&chBName; //get body name from ViconGetBodyName()
float x=0;float y=0;
float z=0;float rx=0;
float ry=0;float rz=0;
float *p_bX=&x;
float *p_bY=&y;
float *p_bZ=&z;
float *p_aX=&rx;
float *p_aY=&ry;
float *p_aZ=&rz;
if(ViconGetBodyAngleAxis(p_BName,p_bX,p_bY,p_bZ,p_aX,p_aY,p_aZ)==true){
cout<<"The name of the body is:"<<chBName<<"\n";
cout<<"The x of the body is:"<<x<<"\n";
cout<<"The y of the body is:"<<y<<"\n";
cout<<"The z of the body is:"<<z<<"\n";
cout<<"The rx of the body is:"<<rx<<"\n";
cout<<"The ry of the body is:"<<ry<<"\n";
cout<<"The rz of the body is:"<<rz<<"\n";
}
else{
cout<<"cannot get body information."<<"\n";}
//*/
//Now you need to write a program to send commands to the draganflyer through PCTx
}
return 0;
}
```

CURRICULUM VITAE

A. CONTACT INFORMATION

WEIZHONG ZHANG

Email:greatzwz@hotmail.com
Phone:0-502-852-0409 (lab)
778 David Fairleigh Ct, #8
Louisville, Kentucky, 40217 USA.

B. EDUCATION

University of Louisville, Louisville, USA
Ph.D., Electrical & Computer Engineering, October, 2009

- Dissertation Topic: "Optimal Trajectory Generation with NTG versus DMOC:Application to an underwater glider and a JPL aerobot",
- GPA: 3.818
- Advisor: Prof. Tamer Inanc, Ph.D

Shanghai Jiaotong University, Shanghai, China
M.Sc., Control Theory and Control Engineering, March, 2005

- Thesis Topic: "'Research on the measurement and control of the main parameters in producing process of polymerized macromolecule'
- Advisor: Prof. Hao Wang, Ph.D

Harbin Engineering University, Harbin, China
B.Sc., Electrical Engineering and Automation, July, 2002

C. RESEARCH EXPERIENCE

University of Louisville, USA

- **Optimal Trajectory Generation for a NASA-JPL aerobot with NTGMay 2008– May 2009**
  In this project, a new aerobot model with consideration of aerodynamics is proposed, optimal trajectories with NTG are generated to shown the solutions are satisfied with the energy and time concerns.

- **Comparisons between NTG and DMOC** **Jan, 2008 – Nov, 2008**
  The project focused on theoretic and practical comparisons between two state of the art optimal trajectory generation methods. NTG is based on B-spline, nonlinear programming and differential flatness while DMOC is dependent on the direct discretization of Lagrange-d'Alembert principle. The pros and cons of both two methods are clarified with application to an underwater glider and a JPL aerobot.
- **Optimal trajectory generation for a glider in time varying 2D B-spline ocean current** **Jan 2007– Jan 2008**
  The ocean current is modeled as time-varying 2D B-spline functions with available sampled ocean current data, minimizing-energy trajectories generated by NTG both for kinematic and dynamic glider are shown consistent with Lagrange Cohere Structures.

Shanghai Jiaotong University, China

- **A New Real-time Method of Measuring PAE Polymerization Degree Jan, 2003 - Oct, 2004**
  A new real-time method for measuring one polymer's polymerization degrees is proposed, the detecting error is less than 3% compared with the off-line chemical time-delay analysis technique. This method can generate the polymer degrees by measuring a few easily-obtained parameters such as flow, temperature and pressures from commercially available instruments. This method is patented and used in Chemical plants in China.

D. JOURNAL PUBLICATIONS

1. **Weizhong Zhang**, Tamer Inanc, Sina Ober-Blőbaum and Jerrold E. Marsden, "Optimal Trajectory Generation in DMOC versus NTG: Application to a Glider," will be submitted to one journal.
2. **Weizhong Zhang**, Tamer Inanc, Alberto Elfes, "Energy Efficient Trajectory Generation for the JPL Aerobot Based on its Decoupled Dynamics", will be submitted to *Journal of Guidance, Control, And Dynamics*
3. Travis Riggs, Tamer Inanc,**Weizhong Zhang**, "The UofL Autonomous Mobile Robotics Systems Testbed," accepted to IEEE Transactions on Control Systems Technology, Dec, 2008.
4. **Weizhong Zhang**, Hao Wang, "Real-time Detecting to Estimate the Average Polymerization Degree of PAE,"*Control and Instruments in Chemical Industry*, 2004 Vol.31 No.6 P.51-53.

E. CONFERENCE PUBLICATIONS

1. **Weizhong Zhang**, Tamer Inanc, Jerrold E. Marsden, "A Tutorial for Applying DMOC to Solve Optimization Control Problems," submitted to the 2010 American Control Conference, Maryland, Jun 30- Jul 2, 2010.
2. **Weizhong Zhang**, Tamer Inanc, Jerrold E. Marsden, "DMOC Approach of Real-Time Trajectory Generation for Mechanical Systems,"in the Proc. of 10th International Conference on Control, Automation, Robotics and

Vision, 17- 20 December 2008, Hanoi, Vietnam.

3. **Weizhong Zhang**, Tamer Inanc, Sina Ober Blőbaum and Jerrold E. Marsden, "Optimal Trajectory Generation for a Dynamic Glider in Ocean Flows Modeled by 3D B-Spline Functions," in the Proc. of the 2008 IEEE International Conference on Robotics and Automation (ICRA08), Pasadena, California, Dec19-23,2008.
4. Travis A Riggs, **Weizhong Zhang**, Tamer Inanc, "The UofL Autonomous Mobile Robotics Systems Testbed," in the Proc of 47th IEEE Conference on Decision and Control (December 2008).

F. HONORS AND AWARDS

1. ICRA 2008 Student Travel Award, May 2008.
2. University Fellowship, University of Louisville, 2005–2007.
3. University of Louisville International Travel Award, Dec 2008
4. UofL Graduate Student Council Travel Award, May 2008
5. UofL NASA-JPL Research Assistantship, 2008.5-2009.10
6. UofL ECE Teaching Assistantship, 2007.9-2008.5

G. SERVICE

- Commissioner, Committee on Diversity and Race Equality, University of Louisville, 2007.9–2008.9.
- Student Member, IEEE Control Systems, Power Engineering, Robotics and Automation Society, 2005.9– Present.
- Co-Chair, Biologically Inspired Robotic in 2008 International Conference on Robotics and Automation, 2008.12.
- President,UofL Chinese Students and Scholars Association, 2006–2007.
- Vice President, UofL Chinese Students and Scholars Association, 2005–2006.
