Lecture Notes in Artificial Intelligence 2466

Subseries of Lecture Notes in Computer Science
Edited by J. G. Carbonell and J. Siekmann

Lecture Notes in Computer Science
Edited by G. Goos, J. Hartmanis, and J. van Leeuwen

Springer
Berlin
Heidelberg
New York
Barcelona
Hong Kong
London
Milan
Paris
Tokyo

Michael Beetz Joachim Hertzberg
Malik Ghallab Martha E. Pollack (Eds.)

Advances in Plan-Based Control of Robotic Agents

International Seminar
Dagstuhl Castle, Germany, October 21-26, 2001
Revised Papers

Springer

Series Editors

Jaime G. Carbonell, Carnegie Mellon University, Pittsburgh, PA, USA
Jörg Siekmann, University of Saarland, Saarbrücken, Germany

Volume Editors

Michael Beetz
Technische Universität München, Institut für Informatik IX
Orleansstr. 34, 81667 München, Germany
E-mail: beetzm@in.tum.de

Joachim Hertzberg
Fraunhofer-Institut, Autonome Intelligente Systeme (AIS)
Schloss Birlinghoven, 53754 Sankt Augustin, Germany
E-mail: hertzberg@ais.fraunhofer.de

Malik Ghallab
LAAS-CNRS, 7, Avenue du Colonel Roche, 31077 Toulouse cedex, France
E-mail: malik@laas-fr

Martha E. Pollack
University of Michigan, College of Engineering
Dept. of Electrical Engineering and Computer Science
Ann Arbor, MI 48019, USA
E-mail: pollackm@eecs.umich.edu

Cataloging-in-Publication Data applied for

A catalog record for this book is available from the Library of Congress

Bibliographic information published by Die Deutsche Bibliothek
Die Deutsche Bibliothek lists this publication in the Deutsche Nationalbibliographie; detailed bibliographic data is available in the Internet at <http://dnb.ddb.de>.

CR Subject Classification (1998): I.2.9, I.2.8, I.2.11, I.2

ISSN 0302-9743
ISBN 3-540-00168-9 Springer-Verlag Berlin Heidelberg New York

Springer-Verlag Berlin Heidelberg New York,
a member of BertelsmannSpringer Science+Business Media GmbH

http://www.springer.de

Printed in Germany

Typesetting: Camera-ready by author, data conversion by Christian Grosche, Hamburg
Printed on acid-free paper SPIN: 10871267 06/3142 5 4 3 2 1 0

Preface

In recent years, autonomous robots, including XAVIER, MARTHA [1], RHINO [2,3], MINERVA, and REMOTE AGENT, have shown impressive performance in long-term demonstrations. In NASA's Deep Space program, for example, an autonomous spacecraft controller, called the REMOTE AGENT [5], has autonomously performed a scientific experiment in space. At Carnegie Mellon University, XAVIER [6], another autonomous mobile robot, navigated through an office environment for more than a year, allowing people to issue navigation commands and monitor their execution via the Internet. In 1998, MINERVA [7] acted for 13 days as a museum tourguide in the Smithsonian Museum, and led several thousand people through an exhibition.

These autonomous robots have in common that they rely on plan-based control in order to achieve better problem-solving competence. In the plan-based approach, robots generate control actions by maintaining and executing a plan that is effective and has a high expected utility with respect to the robots' current goals and beliefs. Plans are robot control programs that a robot can not only execute but also reason about and manipulate [4]. Thus, a plan-based controller is able to manage and adapt the robot's intended course of action — the plan — while executing it and can thereby better achieve complex and changing tasks. The plans used for autonomous robot control are often reactive plans, that is, they specify how the robots are to respond in terms of low-level control actions to continually arriving sensory data in order to accomplish their objectives. The use of plans enables these robots to flexibly interleave complex and interacting tasks, exploit opportunities, quickly plan their courses of action, and, if necessary, revise their intended activities.

The emergence of complete plan-based robot control systems, on the one hand, and the lack of integrated computational models of plan-based control, on the other hand, motivated us to organize a Dagstuhl seminar "Plan-Based Control of Robotic Agents"[1] at Schloss Dagstuhl (21–26 October 2001). The purpose of this seminar was to bring together researchers from different fields in order to promote information exchange and interaction between research groups working on various aspects of plan-based autonomous robot control.

The objective was to identify computational principles that enable autonomous robots to accomplish complex, diverse, and dynamically changing tasks in challenging environments. These principles include plan-based high-level control, probabilistic reasoning, plan transformation, and context and resource-adaptive reasoning. The development of comprehensive and integrated computational models of plan-based control requires us to consider different aspects of plan-based control — plan representation, reasoning, execution, and learning — together and not in isolation. Such integrated approaches enable us to exploit synergies between the different aspects and thereby come up with simpler and more powerful computational models.

[1] For more information on the Seminar see `http://www.dagstuhl.de/01431/`.

This book is a result of this Dagstuhl seminar. It contains a collection of research papers that represent the latest developments in the plan-based control of robotic agents.

Acknowledgments. We wish to thank all authors for their contributions and collaborative effort in producing an up-to-date state-of-the-art book in a very dynamic field. We also thank all other participants in the seminar for their presentations and lively discussions.

We especially appreciated Gerhard K. Kraetzschmar's interest in this project.

We would also like to thank the Dagstuhl organization for offering a wonderful facility, and the Dagstuhl office for their perfect support. Finally, we would like to acknowledge the support provided by the European Network of Excellence in AI Planning (PLANET) and the High-Level Scientific Conference (HLSC) program of the European Union.

July 2002

Michael Beetz
Joachim Hertzberg
Malik Ghallab
Martha Pollack

References

1. R. Alami, S. Fleury, M. Herb, F. Ingrand, and F. Robert. Multi robot cooperation in the Martha project. *IEEE Robotics and Automation Magazine*, 5(1), 1998.
2. M. Beetz, T. Arbuckle, M. Bennewitz, W. Burgard, A. Cremers, D. Fox, H. Grosskreutz, D. Hähnel, and D. Schulz. Integrated plan-based control of autonomous service robots in human environments. *IEEE Intelligent Systems*, 16(5):56–65, 2001.
3. W. Burgard, A.B. Cremers, D. Fox, D. Hähnel, G. Lakemeyer, D. Schulz, W. Steiner, and S. Thrun. Experiences with an interactive museum tour-guide robot. *Artificial Intelligence*, 114(1-2), 2000.
4. D. McDermott. Robot planning. *AI Magazine*, 13(2):55–79, 1992.
5. N. Muscettola, P. Nayak, B. Pell, and B. Williams. Remote Agent: to go boldly where no AI system has gone before. *Artificial Intelligence*, 103(1–2):5–47, 1998.
6. R. Simmons, R. Goodwin, K. Haigh, S. Koenig, J. O'Sullivan, and M. Veloso. Xavier: Experience with a layered robot architecture. *ACM magazine Intelligence*, 1997.
7. S. Thrun, M. Beetz, M. Bennewitz, A.B. Cremers, F. Dellaert, D. Fox, D. Hähnel, C. Rosenberg, N. Roy, J. Schulte, and D. Schulz. Probabilistic algorithms and the interactive museum tour-guide robot Minerva. *International Journal of Robotics Research*, 2000.

Table of Contents

Plan-Based Multi-robot Cooperation 1
Rachid Alami, Silvia Silva da Costa Bothelho

Plan-Based Control for Autonomous Soccer Robots – Preliminary Report . 21
Michael Beetz, Andreas Hofhauser

Reliable Multi-robot Coordination Using Minimal Communication and Neural Prediction 36
Sebastian Buck, Thorsten Schmitt, Michael Beetz

Collaborative Exploration of Unknown Environments with Teams of Mobile Robots 52
Wolfram Burgard, Mark Moors, Frank Schneider

Mental Models for Robot Control 71
Hans-Dieter Burkhard, Joscha Bach, Ralf Berger, Birger Brunswieck, Michael Gollin

Perceptual Anchoring: A Key Concept for Plan Execution in Embedded Systems 89
Silvia Coradeschi, Alessandro Saffiotti

Progressive Planning for Mobile Robots – A Progress Report 106
Lars Karlsson, Tommaso Schiavinotto

Reasoning about Robot Actions: A Model Checking Approach 123
Khaled Ben Lamine, Froduald Kabanza

Lifelong Planning for Mobile Robots 140
Maxim Likhachev, Sven Koenig

Learning How to Combine Sensory-Motor Modalities for a Robust Behavior 157
Benoit Morisset, Malik Ghallab

Execution-Time Plan Management for a Cognitive Orthotic System 179
Martha E. Pollack, Colleen E. McCarthy, Sailesh Ramakrishnan, Ioannis Tsamardinos

Path Planning for Cooperating Robots Using a GA-Fuzzy Approach 193
Dilip Kumar Pratihar, Wolfgang Bibel

Performance of a Distributed Robotic System Using Shared Communication Channels 211
Paul Rybski, Sascha Stoeter, Maria Gini, Dean Hougen, Nikolaos Papanikolopoulos

Use of Cognitive Robotics Logic in a Double Helix Architecture for Autonomous Systems 226
Erik Sandewall

The DD&P Robot Control Architecture – A Preliminary Report 249
Frank Schönherr, Joachim Hertzberg

Decision-Theoretic Control of Planetary Rovers 270
Shlomo Zilberstein, Richard Washington, Daniel S. Bernstein, Abdel-Illah Mouaddib

Author Index 291

Plan-Based Multi-robot Cooperation

Rachid Alami[1] and Silvia Silva da Costa Bothelho[2]

[1] LAAS-CNRS, 7, Avenue du Colonel Roche
31077 Toulouse Cedex 4, France
Rachid.Alami@laas.fr
[2] FURG, Av. Italia Km 8, 96201-000 Rio Grande/RS, Brazil
silviacb@ee.furg.br

Abstract. Several issues arise if one wants to operate a team of autonomous robots to achieve complex missions. The problems range from mission planning taking into account the different robots capabilities to conflict free execution.
This paper presents a general architecture for multi-robot cooperation whose interest stems from its ability to provide a framework for cooperative decisional processes at different levels: mission decomposition and high level plan synthesis, task allocation and task achievement.
This architecture serves as a framework for two cooperative schemes that we have developed: M+NTA for Negotiation for Task Allocation, and M+CTA for Cooperative Task Achievement.
The overall system has been completely implemented and run on various realistic examples. It showed effective ability to endow the robots with adaptive auto-organization at different levels. A number of performance measures have been performed on simulation runs to quantify the relevance of the different cooperative skills that have been proposed.

1 Introduction

Starting from the **Plan-Merging** Paradigm [3] for coordinated resource utilization - and the **M+ Negotiation for Task Allocation Protocol** [8,7] for distributed task allocation, we have developed a generic architecture for multi-robot cooperation [9,6].

This architecture is based on a combination of local individual planning and coordinated decision for incremental plan adaptation to the multi-robot context. It has been designed to cover issues ranging from mission planning for several robots, to effective conflict free execution in a dynamic environment. It is aimed not only to integrate our past contributions but also to allow to investigate new cooperation and coordination schemes.

The goal of this paper, after a brief analysis of related work, is to present an overview of the architecture and to discuss our current instantiation. We will successively address (1) a distributed task allocation protocol and (2) a cooperative task achievement scheme that detects and treats resource conflict situations as well as sources of inefficiency. Finally, we present an implemented system which illustrates, in simulation, the key aspects of our contribution.

M. Beetz et al. (Eds.): Plan-Based Control of Robotic Agents, LNAI 2466, pp. 1–20, 2002.

The overall system allows a set of autonomous robots not only to perform their tasks in a coherent and non-conflict manner but also to cooperatively enhance their task achievement performance taking into account the robots capabilities as well as their execution context.

2 Related Work

The field of multi-robot systems covers today a large spectrum of topics[18,13,28]. We here restrict our analysis to contributions proposing cooperative schemes at the architectural and/or decisional level. In such stream, *behavior-based* and similar approaches [26,25], propose to build sophisticated multi-robot cooperation through the combination of simple (but robust) interaction behaviors. ALLIANCE [27] is a distributed behavior based architecture, which uses mathematically modeled motivations that enable/inhibit behaviors, resulting in tasks (re)allocation and (re)decomposition.

AI-based cooperative systems have proposed domain independent models for agents interaction. For example, [11] and [20] enrich the STRIPS formalism, aiming to build centralized/decentralized conflict-free plans, while [14] develops specialized agents which are responsible for individual plans coordination.

Several generic approaches have been proposed concerning goal decomposition, task allocation and negotiation [4,16]. PGP [19] (and later GPGP [15]) is a specialized mission representation that allows exchanges of plans among the agents. DIPART [29] is a scheme for task (re)allocation based on load balancing. Cooperation has also been treated through negotiation strategies [31] like CNP-based protocols [33], or BDI approaches where agents interaction is based on their commitment to achieve individual/collective goals [22,34]. Another perspective is based on the elaboration of conventions and/or rules. For instance, "social behaviors" [32] have been proposed as a way to program multi-agent systems. In STEAM [35], coordination rules are designed in order to facilitate the cohesion of the group.

Cooperation for achieving independent goals has been mostly addressed in the framework of application-specific techniques such as multi-robot cooperative navigation [36,12,5].

3 A Multi-robot Architecture for Incremental Plan Enhancement

The generic architecture that we propose covers issues ranging from mission planning for several autonomous robots, to effective conflict free execution in a dynamic environment.

This architecture is based on a combination of local individual planning and coordinated decision for incremental plan adaptation to the multi-robot context. It is built on the assumption that, in a complex system composed of several autonomous robots equipped with their own sensors and effectors, the

ability of a given robot, to achieve a given task in a given situation can be best computed using a planner. Indeed, we claim that the robots must be able to plan/refine their respective tasks, taking into account the other robots' plans as planning/refinement constraints, and thus producing plans containing coordinated and cooperative actions that ensure their proper execution and will serve as a basis for negotiation.

It remains to determine what are the relevant decisional problems that should be addressed. The architecture we propose is precisely an answer to this question. It provides a framework where multi-robot decisional issues can be treated at three different levels: the *decomposition* of a mission into tasks (mission planning), the *allocation* of tasks among the available robots and the *tasks achievement* in a multi-robot context (Figure 1).

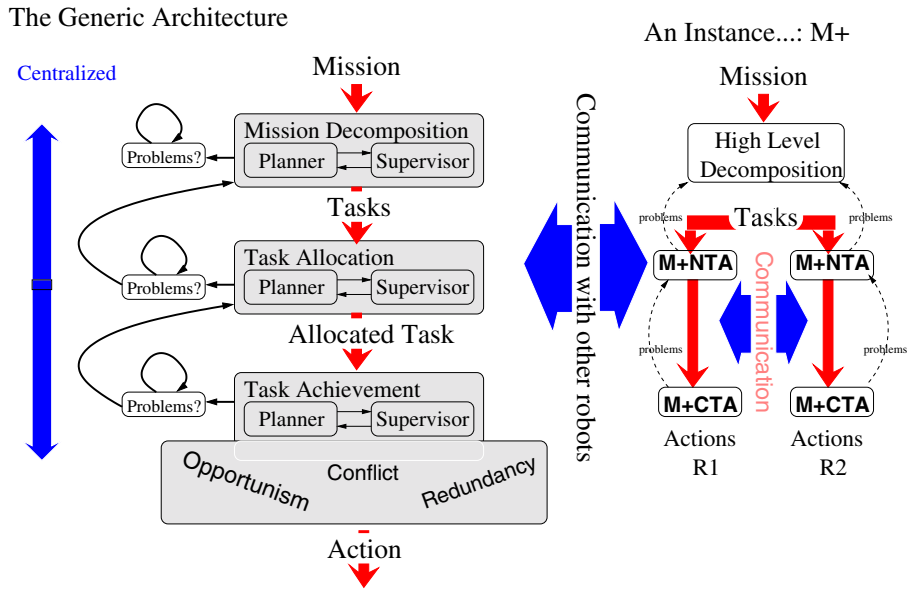


Fig. 1. Our architecture for multi-robot cooperation

Indeed, we claim that it is often possible (and useful) to treat these three issues separately. As we will see, this levels deal with problems of different nature, leading to specific representations, algorithms and protocols.

This architecture is directly derived from the LAAS[1] architecture [1]. It involves a hierarchy of three decisional levels having different temporal constraints and manipulating different data representations. Each level has a reactive (supervisor) and a deliberative component (planner, plan-merger...).

[1] LAAS: LAAS' Architecture for Autonomous Systems.

Communication between robots can take place at a different levels. For a given level, both components communicate with their corresponding component. The reactive components exchange *signals* and run *protocols*; the deliberative components exchange plans, goals and data.

Let us examine the three levels with more detail.

3.1 Mission Decomposition: Planning and Supervision

This is a pure plan synthesis problem. It consists in decomposing a mission, expressed at a very high level, into a set of partially ordered tasks that can be performed by a given team of robots. One can consider that this plan elaboration process is finished when the obtained tasks have a sufficient range and are sufficiently independent to allow a substantial "selfish" robot activity.

We assume that there is no need at this level to know precisely the current robots states. It should be enough to know the types of available robots, their number, their high level features.

An example of such a mission could be transporting and assembling a superstructure in a construction site. It may require to synthesize a sophisticated plan composed of numerous partially ordered tasks to be performed by various robot types with different capabilities: transport of heavy loads, maneuvers in cluttered environment, manipulation...

Mission decomposition is a purely deliberative. It is at this level that there are less needs of context dependent information. It can be done in a central way. It is essentially a one thread process.

Of course it can benefit from several CPUs but this is a distribution of computing load, which is different in nature from problems calling for cooperative decision-making based on independent goals, on various robot capabilities and contexts.

In our current implementation, mission planning is produced by a central high level planner, for instance IxTeT [24], or the mission is provided directly by the user as a set of partially ordered tasks.

3.2 Task Allocation among the Robots

A mission is a set of partially ordered tasks, where each task (T_i) is defined as a set of goals to be achieved. The tasks are allocated to the robots based on their capabilities and on their execution context.

This level is not necessarily distributed. However, its distribution is clearly preferred since task allocation is essentially based on proper or local information. Indeed, the tasks may be allocated (and re-allocated when necessary) incrementally through a negotiation process between robot candidates. This negotiation is combined with a task planning and cost estimation activity which allows each robot to decide its future actions taking into account its current context and task, its own capacities as well as the capacities of the other robots.

We have implemented this level through M+NTA[2]. This system has all necessary protocols and algorithms for cooperative task allocations (see §4).

3.3 Task Achievement in a Multi-robot Context

The allocated tasks, and this is a key aspect in robotics, cannot be directly "executed" but require further refinement taking into account the execution context [1].

Since each robot synthesizes its own detailed plan, we identify two classes of problems related to the distributed nature of the system: (1) coordination to avoid and/or solve conflicts and (2) cooperation to enhance the efficiency of the system. The first class has been often treated in the literature. The second class is newer and raises some interesting cooperative issues linked to the improvement of the global performance by detecting sources of inefficiency and proposing possible enhancements.

Coordination to Avoid Conflicts. Each robot, while seeking to achieve its goal will have to compete for resources, to comply with other robots activities. Indeed, the higher levels, even if they produce valid mission decomposition, do not consider all possible conflicts that may appear at task execution level. We have already treated resource conflict situations as well as coordinated navigation [2,21]. We will see, in the sequel, that the Plan-Merging Paradigm can be extended to more general conflicts.

Cooperation to Enhance the System Performance. We have identified several cooperative issues based on local interactions:

1. **Opportunistic action re-allocation**: one robot can opportunistically detect that it will be beneficial for the global performance if it could perform an action that was originally planned by another robot;
2. **Detection and suppression of redundancy**: it may happen that various robots have planned actions which lead to the same world state. There should be some reasoning capabilities to allow them to decide when and which robot will perform actions that lead to the desired state while avoiding redundant executions;
3. **Incremental/additive actions**: the robots detect that an action originally planned by one robot can be incrementally achieved by several robots with a "cumulative" effect and that this could be beneficial to the global performance.

In our current instantiation of the architecture, M+CTA[3] implements this incremental task achievement level.

[2] NTA: Negotiation for task achievement.
[3] CTA: Cooperative Task Achievement.

3.4 Cooperative Reaction to Contingencies.

The architecture provides hierarchical reaction to contingencies. When a failure (or an unexpected event) occurs at a level, it is first treated at this level and if no solution is found, the higher level is invoked. In the framework of our multi-robot cooperative architecture, this process allows to re-consider the previous allocation or decomposition choices. This should allow a multi-robot team to adapt to its execution context and to *auto-organize* itself in order to perform complex missions in presence of uncertainty.

3.5 Discussion

In the following we discuss some design issues relative to our architecture. Architectural choices may often be considered somehow as arbitrary. Our design is partially intuitive and partially based on our own observations and on the main domains in the literature where multi-robot cooperation has been applied.

For instance, in the great majority of multi-robot systems described in the literature, only one aspect or the other is addressed. But this is only possible if the other aspects are simplified. At the highest level, the mission is often given already decomposed or with a small number of (trivial) decompositions. For example: transferring a bunch of n objects is trivially decomposed in n transfer tasks of individual objects. Numerous other possibilities (perhaps more efficient) may exist depending on the types of objects, the robot capabilities and their current state...

In numerous multi-mobile robot systems, elaborated motion coordination - which clearly belongs to the the task achievement level - is neglected or ignored. Such simplification is acceptable only for non constrained environments where local non-coordinated obstacle avoidance schemes are sufficient.

One, Two, or Three Levels. It may happen that for some applications, it is impossible to separate the mission decomposition and the task allocation aspects because they are too tightly linked. This is the case when the mission decomposition depends heavily not only on the types of robots available in the environment but also on their number and their current situation. In such case, the two levels should be merged in a one step planning process.

The frontier between levels that corresponds to a real qualitative change is between the task allocation and the task achievement levels. But, of course, it is still possible to devise intricate examples that challenge any architectural decomposition.

Cooperative Skills. Not all levels are activated or even present on all robots in a given application. For instance, one can imagine, in a hospital environment, the operation of several teams of mobile robots: a cleaning robots team, a meals and linen delivery team, and a set autonomous wheel-chairs (some of them do not even belong to the hospital)

The cleaning team may cooperate at mission level. The meals and linen delivery team may cooperate at task allocation level. All robots need to cooperate at resource conflict level.

Global Coherence and Efficiency. While the architecture may be considered as satisfactory in terms of identification of the relevant levels of abstractions and their articulation, this is not a guarantee of global coherence nor of efficient operation of the robots.

Indeed, such properties depend primarily on the cooperative schemes and the algorithms that are implemented *inside* each level. For example, the Plan-Merging Paradigm has been devised to provide quite efficient local solutions to most resource conflicts while maintaining two key features [30]:

- The coherence of the global scheme and the ability to detect the situations where it is not applicable;
- A localized management of the planning and coordination processes with, in particularly intricate situations, a progressive transition to more global schemes which may "degrade" to a unique and centralized planning activity.

The following sections present successively M+NTA and M+CTA.

4 M+NTA: Negotiation for Task Allocation

Each robot receives the same mission description, i.e. the same set of partially ordered tasks. At any moment a task is said to be *executable* if all its antecedent tasks are already achieved or under execution. Robots are informed whenever a task is started or finished.

The M+NTA task allocation process allows the robots to incrementally choose a task among the current *executable tasks*. We use an adapted version of the *Contract Net Protocol* [33] for the negotiation. We limit the negotiation and planning to the set of executable tasks because, in general, a plan to perform a task may depend on the state of the world resulting form the previous tasks. The choice criterion will be the costs of the plans elaborated by different robots depending on their capabilities and situations.

Figure 2 shows M+NTA task allocation state diagram. There are 5 possible states: *planning*, *eval-cost*, *candidate*, *best-candidate*, and *idle*.

A robot R_p enters the *eval-cost* state whenever there is an update in the set of *executable tasks* and it is ready to negotiate a new task (1). It invokes its planner (2) in order to synthesize plans for the executable tasks and to estimate their costs (3). Then, R_p selects the task for which it can propose a better cost than the cost announced by other robots, if any.

If there is no such task, R_p enters the *idle* state (8).

If a task T_k is selected, R_p enters the *candidate* state (4). R_p sends its offer to current *best candidate* for this task and waits for an answer. If the answer is positive, meaning that the current *best candidate* accepts to transfer T_k to

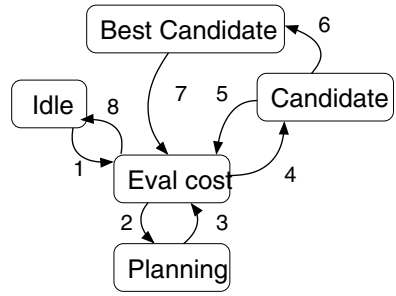


Fig. 2. State diagram for the task allocation protocol

R_p. R_p enters the *best-candidate* state (6). However, the answer can be negative, meaning that the cost of R_p is not better than the cost of the current *best candidate*. R_p then abandons T_k and enters the *eval-cost* state in order to select a new task (5).

When a robot R_p enters the *best-candidate* state for a given task T_k, it holds in this state until either it begins T_k execution or until it receives a better offer from another robot, or until it abandons T_k due to a failure or to a cooperative reaction (7). It then enters the *eval-cost* state in order to select a new task.

M+ Cooperative Reaction

The `M+NTA` scheme also provides a treatment for cooperative robot behavior in case of execution failure. When a problem occurs that prevents a robot R_p from achieving a task T_k, it first tries to re-plan in order to find another set of actions to achieve T_k starting from the new state resulting from the failure. But if R_p does not find a new plan, it sends relevant information with a *request for help* to the other robots and waits. If several robots propose their help, R_p selects the best offer. R_p abandons T_k only if it receives no help offer.

5 M+ Cooperative Task Achievement (M+CTA) and the *Mechanism* Concept

In `M+CTA`. the task achievement level is based on an incremental plan validation process. Starting from a task that has been allocated to it, a robot R_p plans its own sequence of actions, called *individual plan*. This plan is produced without taking into account the other robots' plan. After this planning step, R_p negotiates with the other robots in order to incrementally adapt its plan in the multi-robot context.

A number of conflict/cooperative situation problems are raised when a group of agents share the common use of some entities or devices in the environment.

The *mechanisms* provide a suitable framework for robot cooperation. Indeed, there are numerous applications and particularly for servicing tasks, where the robots often need to operate or to interact with automatic machines or passive devices in order to reach their goals or to satisfy some intermediate sub-goals that allow them to finally reach their main goals. For example, a robot has to open a door in order to enter a room, or heat the oven to a given temperature before cooking a cake, etc..

The *mechanism* can be seen as an extension of the concept of resource: a robot not only allocates and frees a mechanism, it not only consumes or produces it, it can also explicitly manipulate it or act on it, directly or through requests to a controller attached to the mechanism.

The simplest entity that will be dealt with through a *mechanism* is a spatial resource that can be used by only one robot at a time: a place where to park. A door is a little more sophisticated. It may have several states, it may been open or closed, or open to a certain extent. A door can be automatic or manual. Besides, depending on the context, a door should be maintained closed as much as possible or not. Note also that there often exist procedures to operate some machines with several steps and rules to share their utilization. An interesting example is the elevator.

The *mechanisms* allow: (1) to identify the entities of common use, (2) to fix rules to guarantee correct and coherent cooperative *utilization* of such entities and (3) to negotiate their common use among the agents.

5.1 A Scenario of Cooperation

A *mechanism* is a data structure that defines how to use a device or a machine. It defines, somehow, the instructions (or directions) for use: the possible sequences of operations, in what conditions it can be shared or used simultaneously by several users, etc.

In the current version of our system, this knowledge is represented by (see figure 3):

- Known initial and final states,
- A set of alternative *paths*; each path is partially instantiated and represents a valid sequence of actions and state changes of the associated entity.
- A set of *social rules*.

Social Rules impose constraints that must be taken into account during the *mechanisms* use. They have been introduced in order to allow the robots produce easily *merge-able* plans. Social rules specify forbidden or undesirable states and propose some desired states. This information is used by the planner in order to avoid the violation of the rule. Thus, social rules have the following generic description:

$$\textsc{Rule}(type, violation\_state, s, proposed\_state)$$

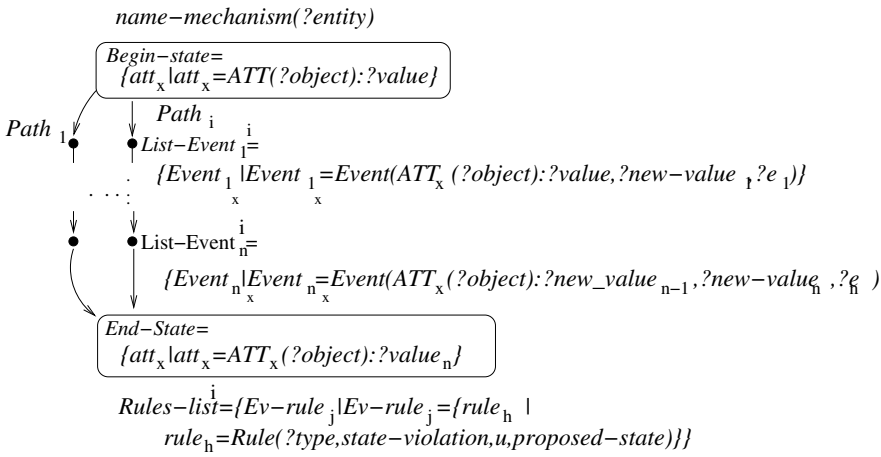


Fig. 3. A generic mechanism M

Social rules are domain dependent; the current version of our system deals with three types of constraints:

1. `amount`: where $violation\_state = (att(?object) : v)$ represents a resource that should limited to a maximum number of s agents. Note that such rules allow to describe the resource constraints of the system. For instance *a limitation of 2 robots at desk D1* can be represented by RULE(`amount`, $(pos\_robot(?r) : D1), 2,$ `OPEN_AREA`), where it is proposed to send the robot to an `OPEN_AREA`, in order to satisfy the rule.
2. `end`: where *proposed_state* must be satisfied at the end of each utilization of the resource. This class guarantees a known final state, allowing the planner to predict the state of an attribute (initial state for the next plan).
3. `time`: where *violation_state* can be maintained true only during a given amount of time s.

The Use of Social Rules in the Planning Phase: We associate to social rules a scalar value called *obligation level*. Whenever a robot plans, it considers the *proposed states* of the rules as mandatory goals that will be added to its current list of goals. However, depending on the obligation level, goals will be posted (1) as a conjunction with the current robot goals or (2) as additional goals that the robot will try to satisfy in subsequent planning steps. In such case, the planner will produce *additional plans* that will achieve each low-level obligation social rule.

During the execution of a plan, the robot may or may not execute these additional plans, thus neglecting temporarily the *proposed state*. Note that if another agent asks the robot to fulfill the *rule proposed state*, it will then (and

only then) perform the associated additional plan. The *obligation level* may also change depending on the context[4].

5.2 *Mechanisms* and *Jobs*

Whenever a robot R_p detects that its plan uses an entity associated with a mechanism M, it builds a *job* M_j^p. A *job* is a dynamic structure, which results from the instantiation of a *path* of a given mechanism in the current robot plan. A job is composed of *steps*. Each *step* has a set of information associated with it: for instance, the agent that effectively executes the action, the other plan actions that depend on it (`successors`), etc. *Jobs* are used as structure and language of negotiation allowing R_p and other agents to decide about the common utilization of an entity. Figure 4 shows a plan produced by robot R_p that uses a furnace. R_p builds a job M_j^p that may be negotiated. This *job* ends when the final state of the associated mechanism is reached.

6 Cooperation Based on *Mechanisms*

The `M+CTA` level involves three activities that correspond to different temporal horizons and may run in parallel: (1) task planning which produces an individual robot plan; (2) the plan negotiation activity which adapts the plan to the multi-robot context; and (3) the effective plan execution.

From time to time, depending on higher level requirements, the robot invokes its own planner and it incrementally appends new sequences of actions to its current individual plan. This is a standard task planning activity; however, the obtained plan satisfies the social rules and is consequently easily *merge-able*.

6.1 Incremental Plan Negotiation

Let us assume that R_p has an individual plan composed of a set of actions A_i^p which manipulate mechanisms. It performs an incremental negotiation process in order to introduce each action A_i^p in the multi-robots context. This operation is "protected" by a mutual exclusion mechanism[5]. The result is a coherent plan which includes all the necessary coordinations and some cooperative actions. It is default free and can be directly executed. However, it remains "negotiable" (other robots can propose a plan modification) until it is incrementally "frozen" in order to be executed. We analyze in the following the different steps involved in this negotiation process.

[4] Note that this notion of social rules is different, or even complementary, from the social behaviors proposed by [32]. While *social behaviors* are explicitly coded in its reactive task execution, the *social rules* are used at the robot decision level as constraints in its planning, negotiation and execution activities.

[5] We assume that the robots are equipped with a reliable inter-robot communication device.

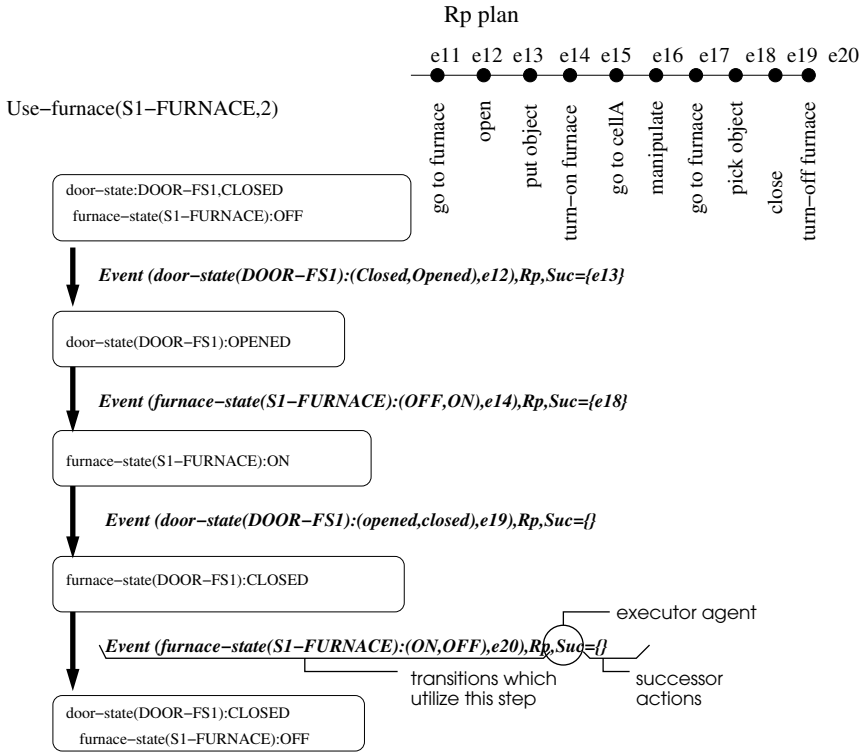


Fig. 4. A *job* corresponding to the use of a furnace by Rp

6.2 The Negotiation Steps:

The negotiation process consists of two steps: the **announcement** and the **deliberation**. During this process, a robot negotiates a set of *jobs* of its current plan[6].

Step 1: The Announcement. Whenever a robot,R_p needs to validate an action A_i^p (belonging to *job* M_j^p. R_p corresponding to the use of a mechanism M), in the multi-robot context, it announces its will to negotiate a job involving M. It obtains the current list of jobs involving M.

Step 2: R_p Deliberates. Having the current job list, R_p has two alternatives associated with its job M_j^p and each member list M_j^q, see figure 5:

[6] We treat together, in one step, all "interleaved" jobs to avoid deadlock situations.

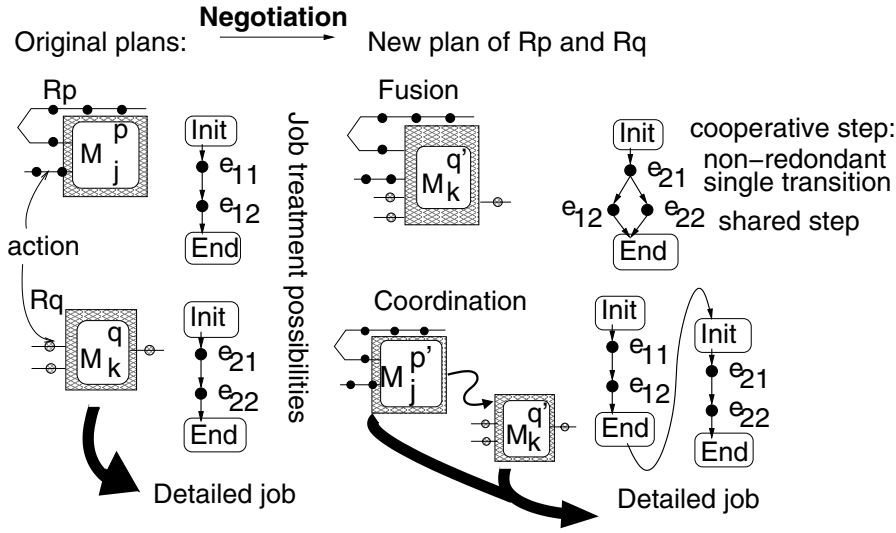


Fig. 5. Job treatment possibilities: fusion or coordination

Fusion: Since our robots are cooperative, the aim is to enhance as much as possible the overall performance. Thus, the robot always try to merge his *job* with the current (already negotiated) *jobs* M_j^q. This is done by trying to detect and treat redundant and shared transitions. The result is a new job $M_j^{q\prime}$ whose actions may be distributed between the different robots.

However, the constraints imposed by *social rules* may prevent a fusion between two jobs. The only remaining solution is to coordinate them in order to avoid conflicts.

Coordination: In this situation R_p can use a mechanism M only after its release by the agents involved in M_j^q. In other words, M_j^p has to be coordinated with M_j^q by adding temporal constraints to the jobs.

After each deliberation process, the robots adapt their plans to the jobs modification. We have defined the following operations:
`insert_message_wait` that introduces a temporal order constraint between two actions belonging to two robots, and `insert`/`delete`, when an action is reassigned to another robot.

Note that such a negotiation process involves only communication and computation and concerns future (short term) robot actions. It can run in parallel with execution of the current coordination plan.

Job Execution Process: Before executing an action A_i^p, the robot **validates** the transition associated to A_i^p. Indeed, a transition remains "negotiable" un-

til its **validation**. Once validated, it is "frozen" and the other robots can only perform insertions after a validated transition. Action execution causes the evolution of the system, resulting in events that will entail new planning, negotiation and execution steps for the robot itself and for the other robots.

7 Illustration

M+ is a complete multi-robot decisional system. It is an instance of the general architecture described in §3. In this implementation, we use PROPICE-PLAN [17] together with a STRIPS-like planner called IPP[23] and a motion planner. Each robot control system runs on an independent workstation which communicates with the other workstations through TCP/IP. For a given application, the environment topology, the robot features, the list of mechanisms and the set of STRIPS actions are input parameters for M+.

We have implemented a first version of the overall system and run it on the realistic simulation platform that was initially developed for the Martha project [2]. Below we describe some of the obtained results.

Our objective here is to measure and compare gains that are obtained when the robots are equipped with the cooperative and coordinations skills that we propose. The interested reader may refer to [10] for a set of documented runs where we examine the different negotiation and cooperation steps.

The application domain that we have chosen is a set of mobile robots in a hospital environment. Servicing tasks are items delivery to beds as well as bed cleaning and room preparation. Figures 6 and 7 show the simulated environment and 14 partially ordered tasks: `T0,...T13` and the initial world state description[7].

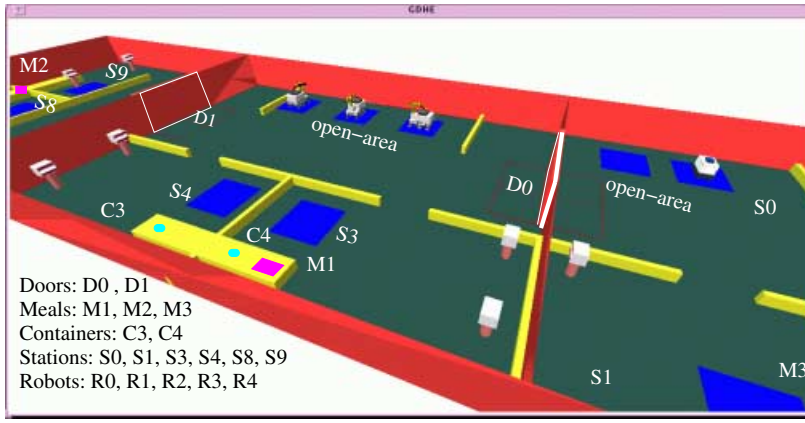


Fig. 6. Example 1: Transfer objects and clean beds in a hospital area

[7] Due to the lack of space, we exhibit here a simplified world state representation.

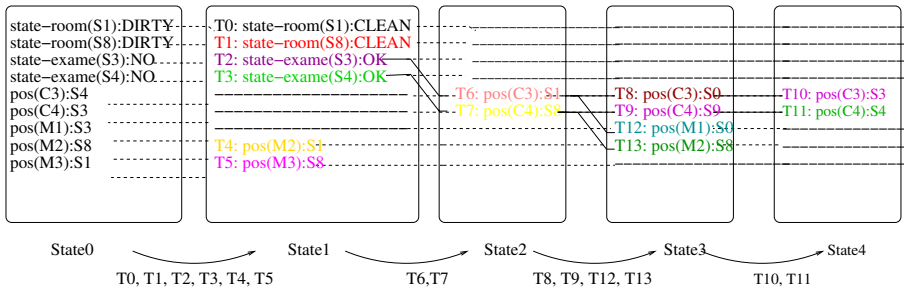


Fig. 7. Example 1: The decomposed mission: 14 individual partially ordered tasks

The robots must negotiate the use of the following *mechanisms*: (1) `clean-room` that allows cleaning actions with cumulative effects when executed several times or by several robots; (2) door-manipulation with `open/close` actions, which can be potentially redundant; and (3) a mechanism that controls the use of the dock station by the robots. This mechanism has an `amount` rule (with low *obligation level*) that limits the number of robots near a station to one.

The set of tasks is transmitted to five robots. After a first phase (described in [8]), they plan and incrementally allocate the tasks using *M+ Cooperative Task Allocation*. Figure 8 shows the individual plans after a number of negotiation processes. Note that `r0` has allocated `T6` in a first step. However it has lost it because `r1` has found a better cost to achieve it. Indeed, `r1` is achieving `T6`. It has elaborated a plan with six actions in order to achieve its main goal `pos(C3):S1` and to satisfy the social rule requiring `state-door(D0):CLOSED` with a high obligation level. Besides, it has also produced an *additional plan* that satisfies rule 1 (with a low obligation level) by introducing a `go-to(OPEN-AREA)` action. After several *jobs* negotiation processes, `r1` deletes its `open` action, which is accomplished by `r3`. This robot opens a first time the door and all robots take advantage of this event. Afterwards, `r1` will close the door for everybody. We can see also the incremental allocation process: while the robots are achieving their current tasks, they try to allocate their future tasks. For instance: `r1-T6` and `r2-T9`.

The overall process continues; the tasks are incrementally planned, negotiated and executed. Figure 9 shows the final result of this run. One can notice, that the robots have satisfied the *social rule* associated to the robot position near the stations. Indeed, some robots detected and deleted redundant actions (open/close door) accomplished opportunistically by others. Besides, some robots also helped the others to clean rooms.

Figure 10 shows the time sharing among execution and deliberation activities. Deliberation activities are decomposed into task allocation and *mechanisms* negotiation. All activities run in parallel. Note that execution activities are more expensive, however `r0` has a high task allocation activity due to the mission na-

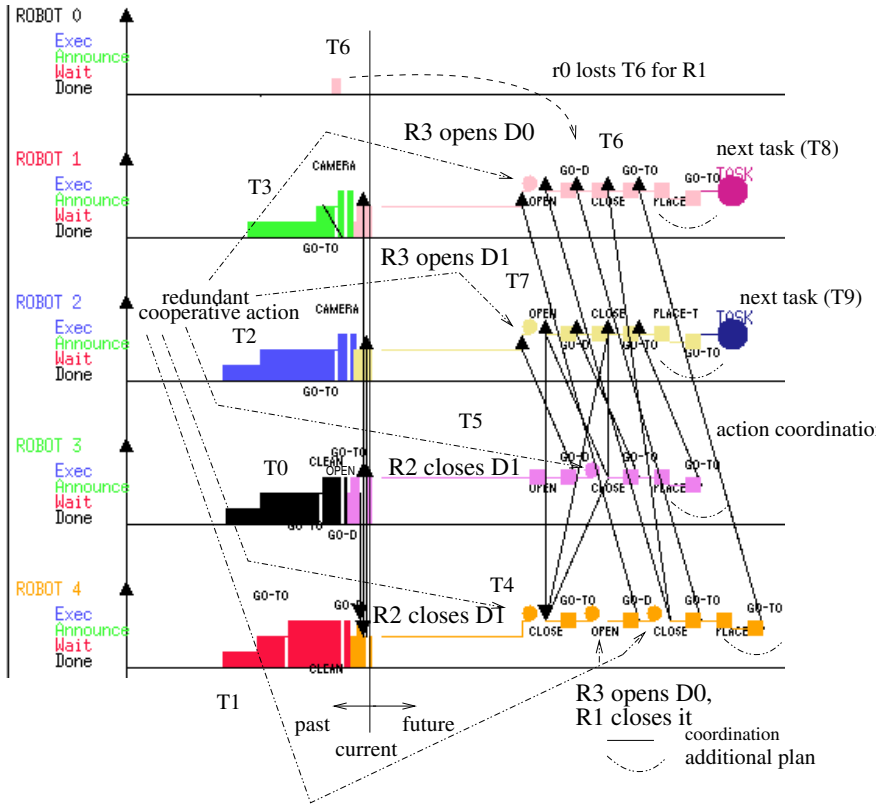


Fig. 8. M+ task achievement process: 5 execution streams corresponding to 5 robots that plan, negotiate and coordinate incrementally their activities. The arrows between robot plans illustrate the temporal constraints induced by the coordination between jobs

ture and to its proper context: the tasks order limits their execution in parallel and `r0` spends a lot of time searching for a task to perform.

We have run the system several times with different parameter values. These parameters are associated with two aspects: the type of cooperation and the number of robots. We have run the system with three different cooperation strategies: (1) `COOP-TOTAL`: treating redundancy and opportunistic incremental help between *jobs*; (2) `NO-INC`: only treating redundant cases with no incremental help; and (3) `NO-COOP`: the system allows only coordination between *jobs*.

On the whole, `COOP-TOTAL` enhances the system performance with better costs and less actions (see Figures 11).

When we change the number of robots, we observe (Figure 12) that the number of achieved actions with 5 and 3 robots is smaller than with 2 robots.

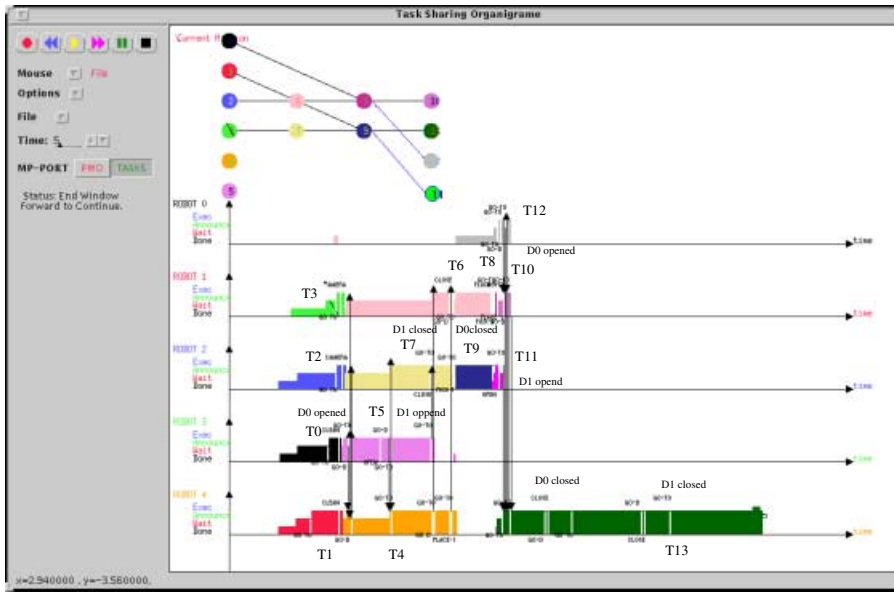


Fig. 9. The final result of the run. All streams are now finished. The top part of the figures shows the partial order between tasks

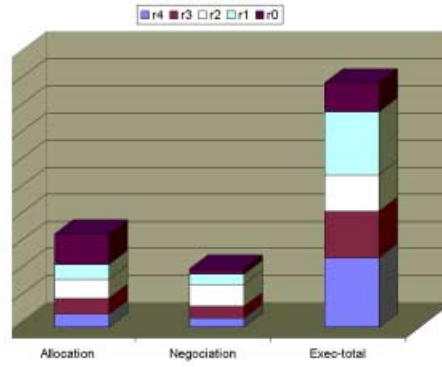


Fig. 10. Time spent in decisional (allocation, negotiation) and execution activities by the different robots

Note that there is no difference between 3 and 5 robots tests; this is due to the nature of mission. The partial order of tasks prevents an optimum deployment of more than 3 robots.

Concerning the the workload, we can see that when we have 5 robots, one of them (`r0`) is almost idle (Figure 12). This fact explains the similar results

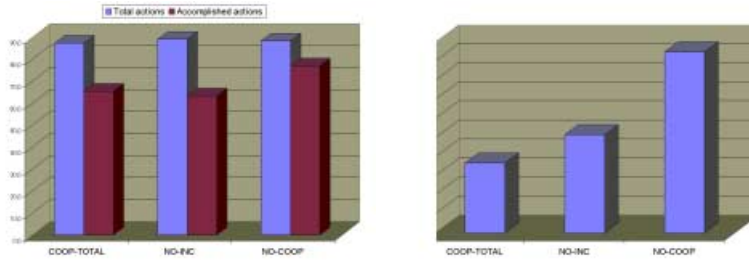


Fig. 11. Relevance of the proposed cooperative schemes. When the robots use all the proposed schemes (COOP-TOTAL), they perform less actions and the global cost is lower

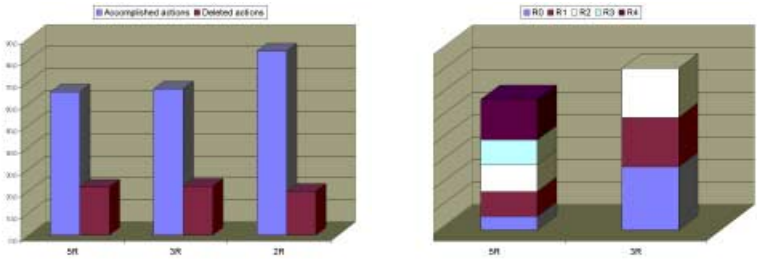


Fig. 12. Illustration of the influence of the number of robots for the same mission. Left: Number of robots vs. planned and achieved actions. Right: Workload for each robot

between 3 or 5 robots tests. However, note that our system has found a very good balance when only three robots are involved.

8 Conclusion

We have proposed a generic architecture for multi-robot cooperation. Its interest stems from its ability to provide a framework for cooperative decisional processes at different levels: mission decomposition and high level plan synthesis, task allocation and task achievement.

We have built an instance of this architecture with negotiation for task allocation and cooperative plan coordination and enhancement at the task achievement level.

We have also discussed a scheme for cooperative multi-robot task achievement, called *mechanism*. This scheme is a key component of our general architecture for multi-robot cooperation. Its main originality comes from its ability to allow the robots to detect and treat - in a distributed and cooperative manner - resource conflict situations as well as sources of inefficiency among the robots.

This architecture has been completely implemented and run on various realistic examples. It showed effective ability to endow the robots with adaptive auto-organization at different levels. We have made some preliminary measures that allow to verify and to quantify the relevance of the different cooperative skills that have been proposed.

It remains to validate this approach through a number of significant different application domains. Besides, we would like to extend and further formalize the overall system and its representational and algorithmic ingredients, taking into account cost and time issues to help planning and negotiation activities.

Besides, it is interesting to observe that this study has raised several issues that deserve further investigations: 1) the opportunistic help for global performance improvement, 2) a class of cooperative issues that can be translated into operations on plans, 3) the integration of a behavior model like the social rules at the planning level in order to synthesize easily merge-able plans.

References

1. R. Alami, R. Chatila, S. Fleury, M. Ghallab, and F. Ingrand. An Architecture for Autonomy. *International Journal of Robotics Research*, 17(4):315–337, April 1998.
2. R. Alami, S. Fleury, M. Herrb, F. Ingrand, and F. Robert. Multi-robot cooperation in the martha project. *IEEE Robotics and Automation Magazine, Special Issues: Robotics and Automation in Europe*, 1997.
3. R. Alami, F. Ingrand, and S. Qutub. A scheme for coordinating multi-robot planning activities and plans execution. In *ECAI'98*, 1998.
4. H. Asama and K. Ozaki. Negotiation between multiple mobile robots and an environment manager. In *IEEE ICRA'91*, pages 533–5382, 1991.
5. K. Azarm and G. Schmidt. A decentralized approach for the conflict-free motion of multiple mobile robots. *Advanced Robotics*, 11(4):323–340, 1997.
6. S. Botelho. *Une architecture décisionnele pour la coopération multi-robots.* PhD thesis, LAAS-CNRS, 2000.
7. S.S.C. Botelho. A distributed scheme for task planning and negotiation in multi-robot systems. In *ECAI'98*, 1998.
8. S.S.C. Botelho and R. Alami. M+: a scheme for multi-robot cooperation through negotiated task allocation and achievement. In *IEEE ICRA'99*, 1999.
9. S.S.C. Botelho and R. Alami. M+: A scheme for multi-robot cooperation through negotiated task allocation and achievement. In *IEEE Int. Conf. on Robotics and Automation (ICRA 2000)*, 2000.
10. S.S.C. Botelho and R. Alami. Robots that cooperatively enhance their plans. In *Distributed Autonomous Robotic Systems 4, Lynne E. Parker, George Bekey, and Jacob Barhen (eds.), Springer*, 2000.
11. C. Boutilier and R. Brafman Planning with concurrent interaction actions. In *AAAI'97*, 1997.
12. A. Brumitt and B. Stentz. Dynamic mission planning for multiple mobile robots. In *IEEE ICRA'96*, 1996.
13. Y. Cao, A. Fukuna, and A. Kahng. Cooperative mobile robotics: Antecedents and directions. *Autonomous Robots*, 4:7–27, 1997.
14. B. Clement and E. Durfee. Top-down search for coordinating the hierarchical plans of multiple agents. In *Third International Conference on Autonomous Agents*, pages 252–259. Association of Computing Machinery, 1999.

15. K. Decker and V. Lesser. Generalizing the partial global planning algorithm. In *Int Journal of Cooperative Information Systems'92*, 1992.
16. M. DesJardins, E. Durfee, C. Ortiz, and M. Wolverton. A survey of research in distributed, continual planning. *AI Magazine*, pages 13–22, 1999.
17. O. Despouys and F. Ingrand. Propice-plan: Toward a unified framework for planning and execution. In *ECP'99*, 1999.
18. G Dudek. A taxonomy for multi-agent robotics. *Autonomous Robots*, 3:375–397, 1997.
19. E. Durfee and V. Lesser. Using partial global plans to coordinate distributed problem solvers. In *IJCAI'87*, 1987.
20. E. Ephrati, M. Perry, and J.S. Rosenschein. Plan execution motivation in multi-agent systems. In *AIPS*, 1994.
21. F. Gravot and R. Alami. An extension of the plan-merging paradigm for multi-robot coordination. In *IEEE International Conference on Robotics and Automation, Seoul, Korea*, May 2001.
22. N. Jennings. Controlling cooperative problem solving in industrial multi-agent systems using joint intentions. *Artificial Intelligence*, 75, 1995.
23. J. Koehler, B. Nebel, J. Hoffmann, and Y. Dimopoulos Extending planning graphs to an adl subset. In *ECP'97*, 1997.
24. P. Laborie. *IxTeT: une approche integrée pour la Gestion de Ressources et la Synthèse de Plans.* PhD thesis, Ecole Nationale Supérieure des Télécommunications, 1995.
25. D. Mackenzie and R. Arkin. Multiagent mission and execution. *Autonomous Robots*, 4:29–52, 1997.
26. M. Mataric. *Interaction and Intelligent Behavior.* PhD thesis, Massachusetts Institute of Technology, 1994.
27. L. Parker. Alliance: An architecture for fault tolerant multirobot cooperation. *IEEE Trans. on Robotics and Automation*, 14(2):220–239, 1998.
28. L. Parker. Current state of the art in distributed robot systems. In *Distributed Autonomous Robotic Systems 4, Lynne E. Parker, George Bekey, and Jacob Barhen (eds.), Springer*, pages 3–12, 2000.
29. M.E. Pollack. Planning in dynamic enviroments: The dipart system. *Advanced Planning Technology: Technology Achievements of the ARPA/Rome Laboratory Planning Initiative*, 1996.
30. S. Qutub, R. Alami, and F. Ingrand. How to solve deadlock situations within the plan-merging paradigm for multi-robot cooperation. In *IEEE IROS'97*, 1997.
31. J.S. Rosenschein and G. Zlotkin. Rules of and encounter: Designing convention for automated negotiation among computers. *Artificial Intelligence - MIT press*, 1994.
32. Y. Shoham and M. Tennenholtz. On social laws for artificial agent societies: Off-line design. *Artificial Intelligence*, 0(75):231–252, 1995.
33. R. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, c-29(12), 1980.
34. G Sullivan, A. Glass, B. Grosz, and S. Kraus. Intention reconciliation in the context of teamwork: An initial empirical investigation. *Cooperative Information Agents III, Lecture Notes in Artificial Intelligence*, 1652:138–151, 1999.
35. M. Tambe. Agent architectures for flexible, practical teamwork. In *First International Conference on Autonomous Agents*, 1998.
36. S. Yuta and S. Premvuti. Coordination autonomous and centralized decision making to achieve cooperative behaviors between multiple mobile robots. In *IEEE IROS'92*, 1992.

Plan-Based Control for Autonomous Soccer Robots

Preliminary Report

Michael Beetz and Andreas Hofhauser

Munich University of Technology
Department of Computer Science IX
Orleanstr. 34, D-81667 Munich, Germany

Abstract. Robotic soccer has become a standard "real-world" testbed for autonomous robot control. This paper presents our current views on the use of plan-based control mechanisms for autonomous robotic soccer. We argue that plan-based control will enable autonomous soccer playing robots to better perform sophisticated and fine tuned soccer plays. We present and discuss some of the plan representations that we have developed for robotic soccer. Finally, we outline extensions of our plan representation language that allow for the explicit and transparent specification of learning problems within plans. This extended language enables robot controllers to employ learning subplans that can be reasoned about and manipulated.

1 Introduction

Controlling autonomous robots entails specifying how they are to react to sensory input in order to accomplish their goals. AI-based robotics researches specializations of this control problem. One of these specializations' key challenges is the creation of an autonomous robot which can accomplish prolonged, complex and dynamically changing tasks while operating in real-world environments. Robotic soccer has become a standard "real-world" testbed for autonomous robot control that exhibits these characteristics [SAB+00].

In robot soccer (mid-size league) two teams of four autonomous robots — one goal keeper and three field players — play soccer against each other. The soccer field is four by nine meters big surrounded by walls. The key characteristics of mid-size robot soccer is that the robots are completely autonomous. Consequently, all sensing and all action selection is done on-board of the individual robots. Skillful play requires our robots to recognize objects, such as other robots, field lines, and goals, and even entire game situations. The robots also need to collaborate by coordinating and synchronizing their actions to achieve their objectives — winning games.

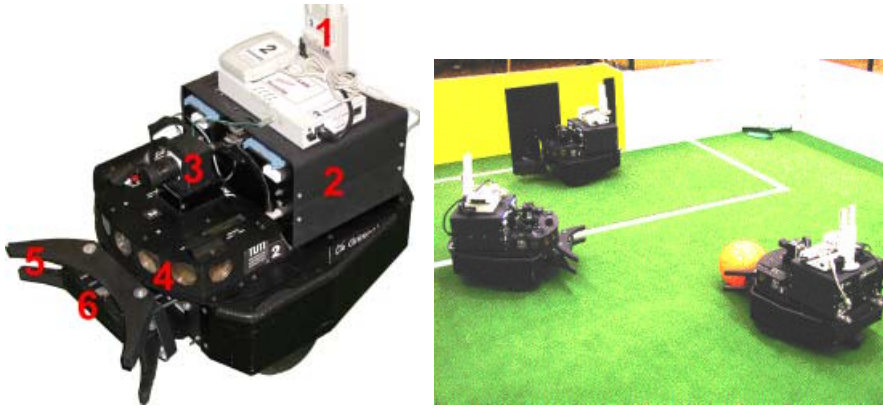
In our research we investigate computational principles enabling autonomous robots to play successfully robotic soccer [BBH+01]. In this paper we focus on a particular subset of these principles: the plan-based control of robotic soccer

M. Beetz et al. (Eds.): Plan-Based Control of Robotic Agents, LNAI 2466, pp. 21–35, 2002.

players. An autonomous robot performs plan-based control if parts of its control program are represented explicitly such that the robot's controller can reason about and manipulate these parts. In the context of robotic soccer the advantages of plan-based control are (1) that a plan for a particular play contains the intentions of the teammates and (2) that experience can be compiled into the plans and reactive execution.

The contributions of this preliminary report on the application of plan-based control techniques to the robotic soccer application are the following ones. First, we present concurrent reactive multi-robot plans for performing complex plays in robotic soccer. Second, we describe and discuss how these plans can be designed to allow for a reliable execution of plays and a smooth and proper integration of plays in the default playing behavior. The plan schemata are designed such that they are reactive, general, and transparent. Third, we design plans with qualitative/functional locations; and, unlike many other plan-based robot applications, sophisticated control of movements.

In the remainder of this paper we proceed as follows. Section 2 describes the robotic soccer players and the main software components of their controllers. Section 3 outlines our computational model of plan-based control in robot soccer. The plan representations used for soccer playing are detailed in Section 4. We conclude with laying out our research agenda for plan-based control in robot soccer and a discussion of related work.



(a) (b)

Fig. 1. An AGILO soccer robot (a) and a game situation (b)

2 The AGILO RoboCup Team

The AGILO RoboCup team is realized using inexpensive, off-the-shelf, easily extensible hardware components and a standard software environment. The team

consists of four Pioneer I robots; one of them is depicted in Figure 1(a). The robot is equipped with a single on-board Linux computer (2), a wireless Ethernet (1) for communication, and several sonar sensors (4) for collision avoidance. A color CCD camera with an opening angle of 90^o (3) is mounted fix on the robot. The robot also has a guide rail (5) and a kicking device (6) that enable the robot to dribble and shoot the ball.

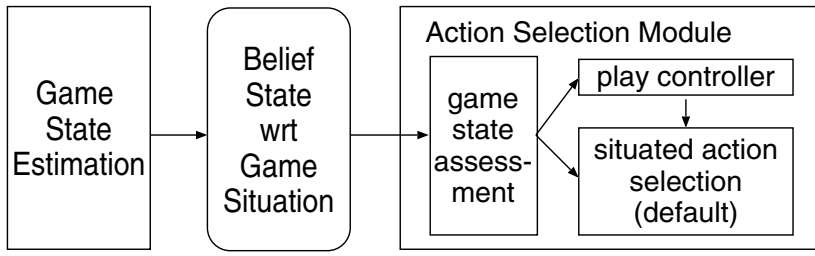


Fig. 2. Software architecture of an AGILO robot controller

The main software components of the AGILO robot controllers are depicted in Figure 2. The incoming video streams are processed by the *vision-based cooperative state estimation module*, which computes the *belief state* of the robot with respect to the game situation, which for now comprise the the estimated positions of the teammates, opponents, and the ball. The action selection module then computes an abstract feature description of the estimated game state that can be used to recognize relevant game situations. There are two basic components for action selection. First, the situated action selection module selects action based on a limited horizon utility assessment. Second, a plan-based controller that enables the team to execute more complex and learned plays. The remainder of this section details the software design and the operation of these software components.

2.1 Cooperative Game State Estimation

Every robot of the AGILO team is equipped with a game state estimation module. Given a video stream captured by the robot's camera and information broadcasted by the other robots of the team, the game state estimation module estimates the robot's own pose and the positions of the ball and opponent robots.

Technically, the module is implemented based on a probabilistic, vision-based state estimation method [SHBB01]. This method enables the AGILO team to estimate the joint positions of the team members on the field and track the positions of independently moving other objects. All poses and positions of the state estimation module contain a description of uncertainty. The state estimators of different robots cooperate to increase the accuracy and reliability of the estimation process. This cooperation between the robots enables them to track

temporarily occluded objects and to faster recover their positions after they have lost track of them. A detailed description of the self-localization algorithm can be found in [HS00] and the algorithms used for cooperative multi-object tracking are explained in [SBHB02].

The properties of the vision-based cooperative state estimation method that are important for the subject of this paper are the following ones. Every robot estimates the complete game situation based on its own sensing data and information broadcasted by the other robots. If a robot does not get information from its teammates its estimate of the game situation becomes more uncertain and inaccurate. Vision-based state estimation is inherently inaccurate. Even small camera vibrations produce very inaccurate distance estimations for robots that are several meters away. Such inaccuracies complicate the fusion of the observations of different robots tremendously. As a result, the robots will sometimes overlook and sometimes hallucinate the robots from the other team.

2.2 The Situated Action Selection Module

Throughout the game the AGILO robots have a fixed set of tasks with different priorities. The tasks are *shoot the ball into the goal*, *dribble the ball towards the goal*, *look for the ball*, *block the way to the goal*, *get the ball*, ... The situated action selection module enables the robot to select a task and to carry out the task such that in conjunction with the actions of the teammates it will advance the team's objectives the most. We consider a task to be the intention of the AGILO robot team to perform certain actions. Action selection and execution is constrained by (1) tasks being achievable only if certain conditions hold (e.g., the robot has the ball) and (2) a robot being able to only execute one action at a time.

We consider a task assignment a_1 to be better than a_2 if there exists a task in a_2 that has lower priority than all the ones in a_1 or if they achieve the same tasks but there exists a task t in a_1 such that all tasks with higher priority are performed at least as fast as in a_2 and t is achieved faster by a_1 than by a_2. This performance criterion implies that if an AGILO robot can shoot a goal it will always try because this is the task with the highest priority. Also, if the AGILO team can get to the ball it tries to get there with the robot that is (under idealized conditions) predicted to reach the ball fastest. This strategy might not yield optimal assignments but guarantees that the most important tasks are achieved as quickly as possible.

3 A Computational Model of Plan-Based Control

While our situated action selection aims at choosing actions that have the highest expected utility in the respective situation it does not take into account a *strategic* assessment of the alternative actions and the respective *intentions* of the teammates. This is the task of plan-based action control. While situated action selection achieves an impressive level of performance it is still hampered by

the requirement for small action and state spaces, a limited temporal horizon, and without explicitly taking the intentions of the teammates into account.

The goal of plan-based control in robotic soccer is therefore to improve the performance of the robot soccer team by adding the capability of learning and executing soccer plays. Soccer plays are properly synchronized, cooperative macro actions that can be executed in certain game contexts and have, in these contexts, a high success rate. Plans for soccer plays specify how the individual players of a team should respond to changing game situations in order to perform the play successfully.

The integration of soccer plays into the game strategies enables robot teams to consider play-specific state spaces for action selection, parameterization, and synchronization. In addition, the state space can reflect the intentions of the other robots. An action that is typically bad might be very good if I know that my teammate intends to make a particular move. Further, action selection can consider a wider time horizon, and the robots can employ play-specific routines for recognizing relevant game situations.

In order to realize an action assessment based on strategic consideration and on considerations of the intentions of the teammates, we develop a robot soccer playbook, a library of plan schemata that specify how to perform individual team plays. The plans, or better plays, are triggered by opportunities, for example, the opponent team leaving one side open. The plays themselves specify highly reactive, conditional, and properly synchronized behavior for the individual players of the team.

The plan-based controller works as follows. It executes as its default strategy the situated action selection. At the same time, the controller continually monitors the estimated game situation in order to detect opportunities for making plays. If an opportunity is detected, the controller decides based on circumstances including the score and the estimated success probability of the intended play whether or not to perform the play.

3.1 Structured Reactive Controllers

The high-level controller of each soccer robot is realized as a structured reactive controller (SRC) [Bee01]. The SRC specifies how each robot robot should respond to sensory input in order to play successful team soccer. SRCs are implemented in RPL (Reactive Plan Language) [McD91]. RPL provides conditionals, loops, program variables, processes, and subroutines as well as high-level constructs (interrupts, monitors) for synchronizing parallel actions. To make plans reactive and robust, it incorporates sensing and monitoring actions, and reactions triggered by observed events.

In a nutshell the structured reactive controller works as follows. The SRC interpreter interprets the structured reactive plan, causing continuous control processes to be activated and deactivated. Threads of control get blocked when they have to wait for certain conditions to become true. For example, if the robot is asked to go to a location L and receive a pass the controller activates the process of moving towards L. The interpretation of the subsequent steps

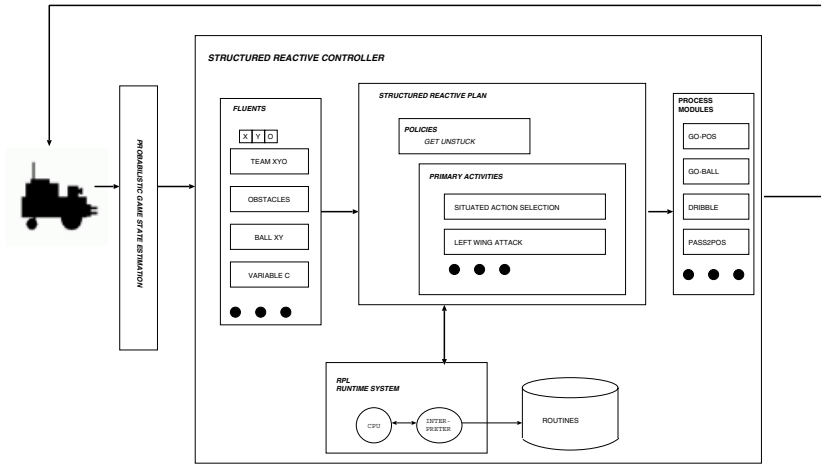


Fig. 3. Architecture of the structured reactive controller (SRC). Components of a structured reactive controller. The structured reactive plan specifies how the robot responds to changes of its fluents. The interpretation of the structured reactive plan results in the activation, parameterization, and deactivation of process modules

are then blocked until the robot has arrived at L (i.e., until the move behavior signals its completion).

Successful soccer play requires robots to respond to events and asynchronously process sensor data and feedback arriving from the control processes. RPL provides *fluents,* registers or program variables that signal changes of their values. Fluents are used to store events, sensor reports and feedback generated by low-level control modules. Moreover, since fluents can be set by sensing processes, physical control routines or by assignment statements, they are also used to trigger and guard the execution of high-level control routines. For example, the soccer SRCs use fluents to store the robot's estimated position, the ones of its teammates, and the ones of recognized opponents. Fluents can also be combined into digital circuits that compute derived events or states such as being in the hallway. For example, a fluent network has the output fluent BALL-THREAT? that is true if an opponent player is closer to the ball than 75 centimeters.

Fluents are best understood in conjunction with the RPL statements that respond to changes of fluent values. The RPL statement whenever F B is an endless loop that executes B whenever the fluent F gets the value "true." Besides whenever, wait for(F) is another control abstraction that makes use of fluents. It blocks a thread of control until F becomes true.

Process Module DRIBBLE(x, y, o, speed, monitors)
1 success condition PLAYER-IS-AT?(PLAYER, X, Y, 20.0)
2 failure condition **ball-lost?**
3 local fluents **dist-ball-fl** ← → C-FLUENT(DIST2BALL)

To facilitate the interaction between high-level play control and continuous control processes, SRCs use the control abstraction *process module*. The process module DRIBBLE, for example, which is provided in the RPL interface for the AGILO robots, is activated with a desired robot pose specified by an x- and y-coordinate and an orientation o, the desired velocity for performing the navigation task, and possible monitors that are supposed to monitor the navigation process. The process module succeeds when the robot has dribbled the ball to the specified destination with a tolerance of twenty centimeters and it fails if the fluent ***ball-lost?*** signals that the ball has been lost while dribbling the ball. The surrounding plans can monitor additional conditions such as the distance of the ball threat, the closest opponent to the ball.

SRCs use control structures for reacting to asynchronous events, coordinating concurrent control processes, and using feedback from control processes to make the behavior robust and efficient. RPL provides several control structures to specify the interactions between concurrent control processes (see the following table). The control structures differ in how they synchronize processes and how they deal with failures.

The in parallel do-construct runs a set of processes in parallel and fails if any of the processes fails. An example use of in parallel do is mapping the robot's environment by navigating through the environment and recording the range sensor data. The second construct, try in parallel, can be used to run alternative methods in parallel. The compound statement succeeds if one of the processes succeeds. Upon success, the remaining processes are terminated. with policy P B, the third control structure, means "execute the primary activity B such that the execution satisfies the policy P." Policies are concurrent processes that run while the primary activity is active and interrupt the primary if necessary. Finally, the plan-statement has the form plan STEPS CONSTRAINTS where CONSTRAINTS have the form order $S_1 \prec S_2$. STEPS are executed in parallel except when they are constrained otherwise. Additional concepts for the synchronization of concurrent processes include semaphores and priorities.

3.2 Situation Assessment

A robot soccer player must be aware of the game situation. Thus an important component of the control software is a module that continually classifies and assesses the respective game situation. This is done by a fluent network used by the SRC. Figure 4 sketches a part of the fluent network used for our current game state estimation.

The input fluents of the networks are the fluents containing the x- and y- coordinates and the orientation of the robot's estimated position, the same fluents for the positions of each teammate, and fluents for the x- and y-coordinates of

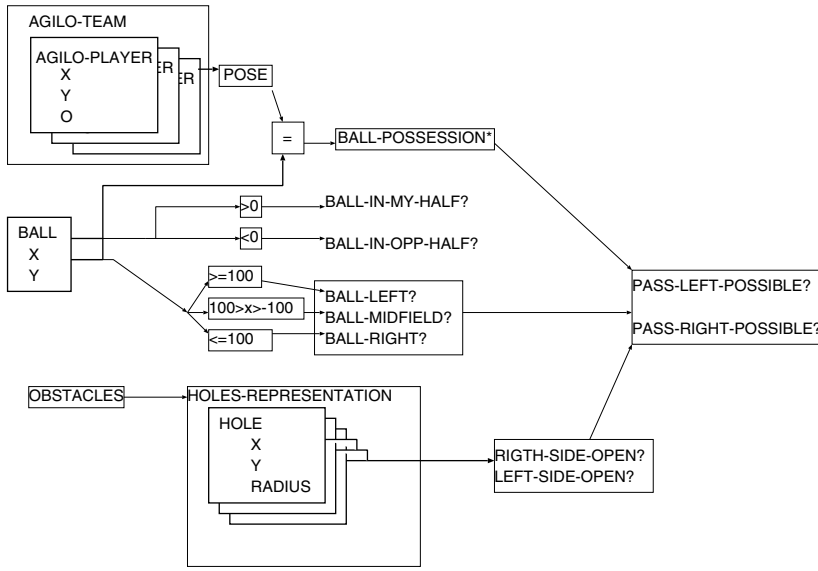


Fig. 4. Fluent network for the assessment of game situations. The input fluents on the left side contain the position estimates for the robot itself, its teammates, the ball, and the opponent players. These fluents are continually updated and their values are propagated through a digital circuit to compute abstract properties of the respective game situation

the ball and the opponent players. The fluent networks compute more abstract features of game situations such as the left or the right side of the field being open, whether there are passing opportunities, whether the team has ball possession, and so on. These abstract features are used to make appropriate decisions about what to do next.

The plan-based controller also creates special purpose fluent networks on demand. For example, if a player is dribbling the ball it creates a fluent that monitors the distance to the closest opponent, the one that is threating the ball.

4 Plan Schemata for Robot Soccer

To see how soccer plays are specified as RPL plans let us consider a specific example that is depicted in Figure 5. In the play the left wing player is supposed to go deep left such that it is free to receive a pass from the play maker in the mid field. Upon receiving the ball the left wing player is to dribble towards the goal and shoot. The play starts with the play maker in ball possession in its own half. The play maker dribbles towards the center of the field and passes the ball deep left. The actions of the two players have to be synchronized such that the

left wing player is already at its destination and ready to receive the pass when the play maker wants to pass.

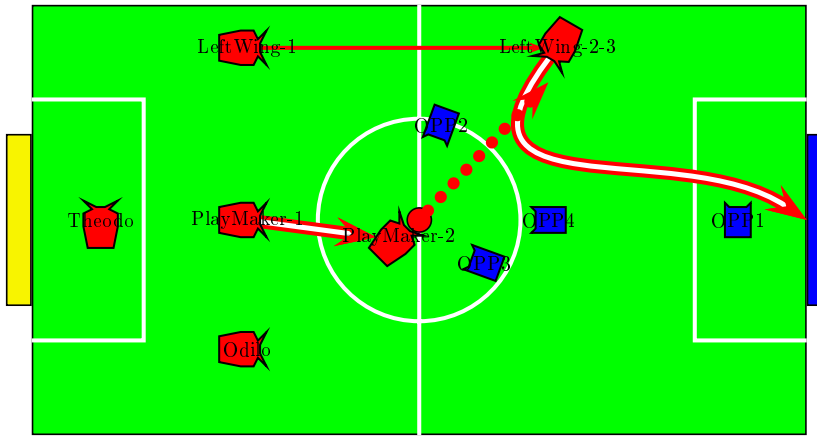


Fig. 5. Example of the soccer play that depicts the left wing attack discussed in the section. Dribblings are indicated by double lines, passes by dotted lines, and running by single lines

Specifying such plays as robot plans requires careful engineering. First, when the team intends to make the play the role of the play maker and the left wing player have to be assigned to the most appropriate robots. The second issue is the specification of the locations where the play maker should pass the ball from and where the left wing player should wait for the ball. Obviously, we should not specify these locations accurately before executing the play. Rather the locations should use fuzzy and qualitative specifications for the positions that are dynamically adapted as the game situation changes during the play. For example, the left wing player should go to a position deep left where the play maker can pass the ball from the center without the ball being in danger of being intercepted. In addition, the position should be such that the left wing player can dribble towards the goal and will have a shooting opportunity. A third issue is that of the proper synchronization between players. In our case it is required that the play maker does not pass before the left wing player is ready to receive the ball. In soccer this timing is even harder because neither the play maker nor the left wing player should be required to wait for the other player to be ready. Such waiting periods would make it easier for the defending team to predict and destroy the intended plays. There are many other issues such as play monitoring and the introduction of opportunistic and recovery moves, which we have not addressed in our research yet.

In order to specify plays modularly, transparently, and compactly we have introduced a particular plan macro called def soccer play. The macro is designed

such that it can capture the kinds of information that soccer coaches typically present on a drawing board when explaining a new play to their players. This information include role assignments, snap shots of the play that display the different stages of the play, and the passes to be played.

```
def soccer play LeftWingAttack1
  triggering condition  ball-possession? ∧  pass-left-possible? ∧  left-side-open?
  failure condition  ball-lost?
  success condition  play-succeeded?
  with play roles LeftWing ← LeftMostPlayer(Agilo)
                  PlayMaker ← BallPossessingPlayer(Agilo)
    with intended passing line ⟨From,To⟩
                               ← BestPassingLine(FieldCenter, DeepLeft)
      steps of LeftWing are
               (1) GoPos(To)
               (2) TurnTo(From)
               (3) ReceiveBall()
               (4) DribbleTowardsGoalAndShoot()
      steps of PlayMaker are
               (1) DribbleTo(From)
               (2) Pass(To)
      synchronizations
        end simultaneously Step(LeftWing,2) and Step(PlayMaker,1)
```

Fig. 6. Specification of a robot soccer play as an RPL plan

The RPL play specification is depicted in Figure 6. The triggering condition specifies those game situations that constitute opportunities for successfully carrying out the LeftWingAttack1. In our case, we consider those game situations as opportunities, in which the Agilo team is in ball possession, the left wing is open, and there is a passing opportunity to the deep left. These conditions are computed by the fluent network shown in Figure 4. The failure and success conditions are also monitored by fluents; in our case the fluent ***ball-lost?*** signals a failure of plan execution and the fluent ***play-succeeded?*** the successful completion of the play. The statement with play roles assigns the appropriate robots for the left wing and the play maker roles. In our case, the left wing attacker role is taken by the left most Agilo player and the play maker role by the robot that is in ball possession.

The last supporting data structure, the intended passing line, is computed by the function Best-Passing-Line. The function takes fuzzy position descriptions for the positions from which (field center) and to which (deep left) the pass should be played. In addition, the function takes additional constraints such as all opponent players not being able to intercept the ball and a heuristic evaluation function for both positions as arguments. The heuristic evaluation function considers conditions such as the deep left position yielding an immediate

scoring opportunity. The body of the play specification formalizes the individual steps to be performed by the left wing attacker and the play maker and the synchronization constraints between them. The left wing attacker first goes to the position where the pass should be played to, then turns towards the position where the pass comes from, then receives the ball, and finally, dribbles towards the goal and shoots the ball if a scoring opportunity presents itself. The play maker is first to dribble to the center and then to pass the ball to the deep left reception position. The additional synchronization constraint tells the two robots that the left wing attacker should be ready for receiving the ball as soon as the play maker is ready to pass.

The plan schema represents the play compactly, transparently, and modularity. It makes the key control decisions explicit. These decisions include how to play the pass, how to synchronize the robots' actions, when to play the pass, how to dribble to the goal, and when to shoot. The explicit representation of these control decisions enables experience based learning mechanisms to make the play more effective by tuning it.

The plans for the different plays are applied by the top-level plan for the offense of the AGILO robots, which looks as follows:

```
def interp proc OFFENSE ()
  with policy whenever  new-assessment-period? ∧ ¬ executing-play?
                        ∧  opportunity-for-attack-1?
                        ∧ ¬ taking-opportunity?
                EXECUTE(LEFTWINGATTACK1)
  with policy whenever  left-wing-attack-active? ∧  play-threat?
                TERMINATE(LEFTWINGATTACK1)
    APPLYACTIONSELECTIONSTRATEGY(DEFAULT)
```

The plan for playing offense executes the default action selection strategy and monitors the game situation for opportunities to perform specific plays and threats that constitute risks for currently executed plays. This is done by two policies. The first one continually checks the situation for play opportunities and finally makes the decision about whether or not to take an opportunity. The second policy monitors the game situation during the execution of a play. Whenever a play threat is detected the play is terminated.

5 Perspectives of Plan-Based Control in Autonomous Robot Soccer

In this section we will outline our research agenda for plan-based control in autonomous robot soccer. We believe that the advantage of having plans, control programs that can be automatically reasoned about and manipulated, is mainly in the development and training phase. At execution time we only use situated and reactive plan execution.

Consider a particular game situation within a particular play, which is admittedly well beyond the competence of current autonomous robotic soccer teams.

In the final phase of a play a robot is supposed to dribble the ball towards the goal and shoot a goal. Programming a control routine that specifies how to dribble towards the goal, when and how to shoot is very difficult and tedious and it is unlikely that such a program will achieve very good performance. In particular, the decision criteria might also depend on the goal keeper strategy of the opponent goal keeper. The acquisition of such a routine is obviously a learning task. Having learned this task the performance might still be not sufficient and therefore we modify the whole play in order to use a teammate as a supporting player. Now our robot has an additional option, namely to pass to the supporting player

Now, two things should happen. First, the learning problem of the first robot should be automatically changed to adopt the additional option to pass the ball. Second, the routine for dribbling and shooting the goal that has been learned from experience should be exploited for the second learning task.

We believe that one viable way of realizing adaptive robots that are capable of doing this, is to implement the controllers of these robots as plan-based controllers and extend the plan representation such that it can explicitly represent learning problems.

Several researchers have started to look into this research direction. Thrun [Thr00] has developed a C-based library that extends the programming language C with probabilistic reasoning and learning capabilities. A number of researchers have investigated different approaches to use partial program specifications for speeding up the process of reinforcement learning [PR98,AR01,BRST00,SPS98].

In our research, we propose $\mathbf{RPL_{learn}}$ as an extension of the reactive plan language RPL (Reactive Plan Language). $\mathrm{RPL_{learn}}$ should enable programmers to implement complex fine tuned control routines efficiently by coupling programming and automatic learning mechanisms. $\mathrm{RPL_{learn}}$ extends RPL by providing three additional components. First, a software library with automatic learning tools. Second, language constructs for embedding learning problems for control tasks into structured reactive controllers, and third, a sublanguage for specifying learning problems.

Figure 7 gives an example of a piece of $\mathrm{RPL_{learn}}$ code. This piece of code contains a declarative description of a learning problem and statements that specify how the learning task is to be incorporated into robot control. The plan piece specifies that a local control problem can be considered as a Markov Decision Problem with a given state and action space, a reward function, and an initial policy. Many aspects of the learning problem, such as the state and the action space, are represented explicitly and can therefore be reasoned about and manipulated.

6 Related Work

Most teams in robotic soccer use general, dynamic game strategies, which aim at selecting situation-based the actions with the highest expected utilities or the actions that are subjectively expected to be the best ones. Such general game

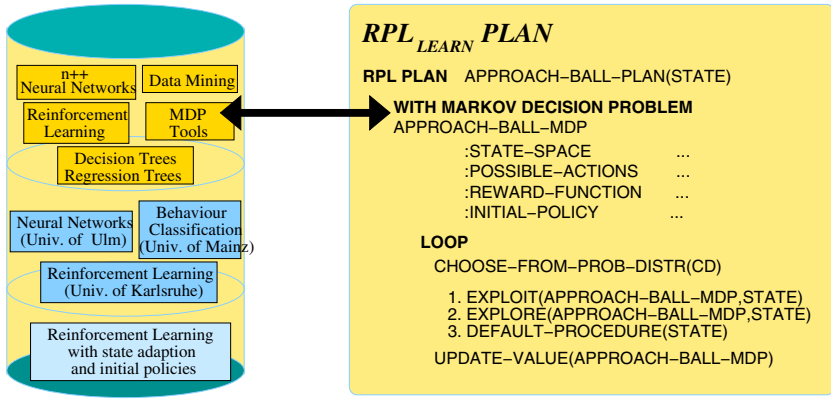


Fig. 7. The figure shows an $\mathbf{RPL_{learn}}$ plan and the library of usable learning tools

strategies comprise *behavior networks* [NW00], *potential fields* [CB97,Bal98], reinforcement learning [SV99b], and prioritized lists of control rules [VSHA98].

Behavior Networks [Mae90] are action selection mechanisms in which the concurrent control routines are activated through activation potentials that are propagated by the respective belief state as well as the respective goals. At any time the behavior with the highest degree of activation is executed. Dorer [Dor99] and Nebel et al. [NW00] remark that the success of this approach in the robocup application crucially depends on a fine grained modeling of the state and action space. The idea of potential fields [Lat91] is the estimation of the utility of actions based on the potential of the resulting situations. In robotic soccer [CB97], the potential field is the sum of component potential fields, which evaluate individual aspects of game situations, such as the distance of the closest teammate to the ball.

In reinforcement learning [SV99b,SV96,RMM$^+$00] soccer games are considered as a finite automata, where the states represent the subjective game situations and the actions cause stochastic transitions in the state space. The goal of the learn process is to compute a mapping from states to actions that have the highest expected utilities in the respective states.

Another method for specifying game behavior is the use of prioritized lists of decision rules [BHKS99] and hierarchically structured rule bases [SV99a].

7 Conclusions

In this paper we have investigated the application of plan-based control techniques to autonomous robot soccer. Unlike other applications such as robot courier applications, tour-guide applications, and spacecraft control, robot soccer confronts plan-based control with distinctive challenges. These challenges in-

clude the control and synchronization of sophisticated movements in adversarial domains, experience-based acquisition of playing skills, qualitative and utility-based parameterization of control routines. We have presented and discussed our prototypical plan schemata for specifying soccer plays.

Our current perception is that plan-based reasoning and learning should be applied in order to achieve faster performance gains in experience-based learning. This, however, remains to be shown in extensive, empirical investigations.

The next steps on our own research agenda are the extension of our plan language. RPL using means for partially specifying game behavior and means for learning how to best replace nondeterministic choices using decision criteria that have been acquired through experience-based learning.

References

AR01. D. Andre and S. Russell. Programmable reinforcement learning agents. In *Proceedings of the 13th Conference on Neural Information Processing Systems*, pages 1019–1025, Cambridge, MA, 2001. MIT Press.

Bal98. T. Balch. Integrating learning with motor schema-based control for a robot soccer team. *Lecture Notes in Computer Science*, 1395:483–492, 1998.

BBH$^+$01. M. Beetz, S. Buck, R. Hanek, T. Schmitt, and B. Radig. The AGILO autonomous robot soccer team: Computational principles, experiences, and perspectives. In *International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS) 2002*, page to appear, Bologna, Italy, 2001.

Bee01. M. Beetz. Structured Reactive Controllers. *Journal of Autonomous Agents and Multi-Agent Systems*, 4:25–55, March/June 2001.

BHKS99. T. Bandlow, R. Hanek, M. Klupsch, and T. Schmitt. Agilo RoboCuppers: RoboCup Team Description. In *Third International Workshop on RoboCup (Robot World Cup Soccer Games and Conferences)*, 1999.

BRST00. C. Boutilier, R. Reiter, M. Soutchanski, and S. Thrun. Decision-theoretic, high-level robot programming in the situation calculus. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, Austin, TX, 2000.

CB97. Thomas R. Collins and Tucker R. Balch. Teaming up: Georgia tech's multi-robot competition teams. In *Proceedings of the 14th National Conference on Artificial Intelligence and 9th Innovative Applications of Artificial Intelligence Conference (AAAI'97/IAAI'97)*, pages 785–786, Menlo Park, July 27–31 1997. AAAI Press.

Dor99. Klaus Dorer. Behavior networks for continuous domains using situation-dependent motivations. In Dean Thomas, editor, *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI'99-Vol2)*, pages 1233–1238, S.F., July 31–August 6 1999. Morgan Kaufmann Publishers.

HS00. R. Hanek and T. Schmitt. Vision-based localization and data fusion in a system of cooperating mobile robots. In *International Conference on Intelligent Robots and Systems (IROS)*, 2000.

Lat91. J.-C. Latombe. *Robot Motion Planning.* Kluwer Academic Publishers, Boston, MA, 1991.

Mae90. P. Maes. How to do the right thing. *Connection Science Journal, Special Issue on Hybrid Systems*, 1, 1990.

McD91. D. McDermott. A Reactive Plan Language. Research Report YALEU/DCS/RR-864, Yale University, 1991.

NW00. B. Nebel and T. Weigel. The CS Freiburg 2000 team. In *4th International Workshop on RoboCup (Robot World Cup Soccer Games and Conferences)*, 2000.

PR98. R. Parr and S. Russell. Reinforcement learning with hierarchies of machines. In Michael I. Jordan, Michael J. Kearns, and Sara A. Solla, editors, *Advances in Neural Information Processing Systems*, volume 10. The MIT Press, 1998.

RMM+00. M. Riedmiller, A. Merke, D. Meier, A. Hoffmann, A. Sinner, O. Thate, Ch. Kill, and R.Ehrmann. Karlsruhe Brainstormers 2000 - A Reinforcement Learning approach to robotic soccer. 2000.

SAB+00. P. Stone, M. Asada, T. Balch, R. D'Andrea, M. Fujita, B. Hengst, G. Kraetzschmar, P. Lima, N. Lau, H. Lund, D. Polani, P. Scerri, S. Tadokoro, T. Weigel, and G. Wyeth. Robocup-2000: The fourth robotic soccer world championships. *AI Magazine*, pages 495–508, 2000.

SBHB02. T. Schmitt, M. Beetz, R. Hanek, and S. Buck. Watch their moves: Applying probabilistic multiple object tracking to autonomous robot soccer. page to appear, Edmonton, Canada, 2002.

SHBB01. T. Schmitt, R. Hanek, S. Buck, and M. Beetz. Cooperative probabilistic state estimation for vision-based autonomous mobile robots. 2001.

SPS98. R. Sutton, D. Precup, and S. Singh. Between MDPs and Semi-MDPs: Learning, planning, and representing knowledge at multiple temporal scales. *Journal of AI Research*, 1998.

SV96. Peter Stone and Manuela Veloso. Collaborative and adversarial learning: A case study in robotic soccer. In Sandip Sen, editor, *Working Notes for the AAAI Symposium on Adaptation, Co-evolution and Learning in Multiagent Systems*, pages 88–92, Stanford University, CA, March 1996.

SV99a. P. Stone and M. Veloso. Task decomposition and dynamic role assignment for real-time strategic teamwork. In Jörg Müller, Munindar P. Singh, and Anand S. Rao (eds.), *Proceedings of the 5th International Workshop on Intelligent Agents V: Agent Theories, Architectures, and Languages (ATAL'98)*, volume 1555 of *LNAI*, pages 293–308, Berlin, July 04–07 1999. Springer.

SV99b. Peter Stone and Manuela Veloso. Team-partitioned, opaque-transition reinforcement learning. In Oren Etzioni, Jörg P. Müller, and Jeffrey M. Bradshaw, editors, *Proceedings of the Third Annual Conference on Autonomous Agents (AGENTS'99)*, pages 206–212, New York, May 1–5 1999. ACM Press.

Thr00. S. Thrun. Towards programming tools for robots that integrate probabilistic computation and learning. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, San Francisco, CA, 2000. IEEE.

VSHA98. M. Veloso, P. Stone, K. Han, and S. Achim. The CMUnited-97 small robot team. *Lecture Notes in Computer Science*, 1395:242–252, 1998.

Reliable Multi-robot Coordination Using Minimal Communication and Neural Prediction

Sebastian Buck, Thorsten Schmitt, and Michael Beetz

Munich, University of Technology, Germany
{buck,schmittt,beetzm}@in.tum.de

Abstract. In many multi-robot applications, such as robot soccer, robot rescue, and exploration, a reliable coordination of robots is required. Robot teams in these applications should therefore be equipped with coordination mechanisms that work robustly despite communication capabilities being corrupted.
In this paper we propose a coordination mechanism in which each robot first computes a global task assignment for the team that minimizes the cost of achieving all tasks, and then executes the task assigned to itself. In this coordination mechanism a robot can infer the intentions of its team mates given their belief states. Lack of information caused by communication failures causes an increase of uncertainty with respect to the belief states of team mates. The cost of task achievement is estimated by a sophisticated temporal projection module that exploits learned dynamical models of the robots. We will show in experiments, both on real and simulated robots, that our coordination mechanism produces well coordinated behavior and that the coherence of task assignments gracefully degrades with communication failures.

1 Introduction

Multiple collaborating robots with a common goal usually have to coordinate their actions in order to avoid physical interferences and to achieve a maximum of speed-up. Reasonably in cooperative multi-robot systems the common goal is decomposed into several tasks related to the individual robots of the system. The decomposition in terms of tasks is unique and changes depending on the current world state. Assuming such tasks can be performed by a single *action* each the question is how to assign tasks (actions) to the robots. Actually a sequence of actions is required for each robot to achieve the common goal. The complexity of this combinatorial problem increases exponential with the number of robots. Moreover the *cost* of a robot executing a certain action must be well known. This requires an accurate prediction.

Typical multi-robot applications dealing with this problem include robot soccer, exploration, mine sweeping and messenger systems. The main advantages of multi-robot systems over single robot systems are speed-up and fault tolerance [9,11]. But without reasonable coordination a multi-robot system can even be less efficient than a single robot system (e.g. two robots block each other). Therefore

M. Beetz et al. (Eds.): Plan-Based Control of Robotic Agents, LNAI 2466, pp. 36–51, 2002.

Mataric [18,19] suggests that control in multi-robot systems must be addressed as a separate, novel, and unified problem, not an additional 'module' within a single-robot approach.

Information acquisition of robots in general occurs by sensors or communication. To achieve consistency and cooperation in multi-robot behavior some kind of common basis for situation assessment is necessary (e.g. synchronization or a common world model) [30]. Young et al. [32] distinguishes between leader-following and behavioral schemes: While in the first approach one robot organizes the whole coordination and assigns actions to the other robots in the latter approach the robots autonomously decide supported by more or less communication. An example for a leader-robot system is given in [10]. While in the leader-following approach success strongly depends on the leader-robot and therefore fault tolerance is poor the behavioral approach requires intelligent software and computational resources on *all* robots. Behavioral systems can be categorized in those with a shared model of the environment, those with a shared abstract description of the environment's state (e.g. at the planning level), and those without shared information. Examples for systems using global maps are the collaborative exploration systems described in [8,27]. Alur et al. [1] periodically exchanges information at discrete time intervals. Many other systems rely on communication too [14,17,23]. The system MAPS [31] uses globally shared information remodeled in an abstract virtual space. multi-robot systems with a shared model of the environment require the computation of such a model which is nontrivial. Systems with synchronization on an abstract description level oblige to change the implementation whensoever anything changes on that abstract level (e.g. priorities in planning). Furthermore it is argued that communication may lead to delays in information acquisition and can increase complexity and degrade multi-robot systems [12,16,26]. However, using no shared information means to be dependent on an accurate state estimation of the environment.

Assuming there is a common basis for action selection that characterizes the current state of the whole multi-robot system there is still the question of how to best coordinate the robots' actions. Agents must reason about expected team utilities of future team states [29]. Forestalling interferences and estimating the team utility requires in general the prediction of the consequences of a particular action assignment for the robot team.

The approach proposed in this paper engages a minimum of communication while primarily relying on a robot's local information. The robots in our system share no global model of the environment but individually estimate the state of the environment with such an accuracy that a periodically exchange of some key features (e.g. distances and angles) of the local estimations among the robots suffices to ensure a cooperative action selection. Moreover these key features enable one robot to put itself in each others robot's situation and thereby to compute the action selected by the other robot. The action selection process employs a state-projection system based on neural networks that is able to project the robots' states into the future. This Projector is supported by a hybrid multi-

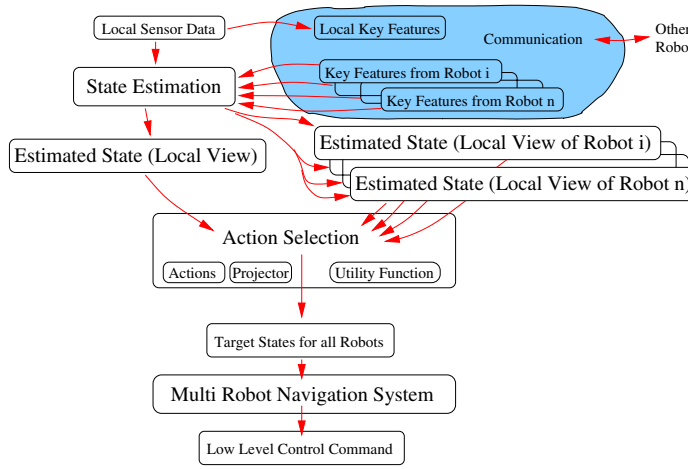


Fig. 1. A robot architecture in a multi-robot system to facilitate cooperative behavior: The robot's *state estimation* is based on *local sensor data* and *key features* from the other robots of the system (if available via communication). Local key features are sent to the other robots. The robot estimates the environment's state not only from its local view but also from the views of the other robots. Thus cooperative *action selection* is done by a *utility function* which can rely on all of the robot's *local views*. A neural projector supports the utility function by estimating the time need for desired state changes. Thereafter the target states of the actions are given to the *multi-robot navigation system* which considers path planning and computes the appropriate *low level control command* for the robot

robot navigation system. The whole system is implemented and tested in highly dynamic robot soccer environment.

The remainder of the Paper is organized as follows: In Section 2 we give an overview on the proposed multi-robot coordination architecture and the action selection algorithm and discuss advantages and disadvantages. Section 3.1 shortly describes how a single robot localizes itself and other objects and generates a model of its environment. Section 3.2 introduces the neural projection system and describes how it works in conjunction with the employed multi-robot navigation system and the action selection algorithm. In Section 4 we put our approach in relation with two concrete examples of robot soccer ((a) which robot is to go for the ball and (b) playing double pass without explicitly learning it) and give results of empirical investigations. Finally in section 5 we close with a conclusion.

2 Overview

To coordinate a team of multiple robots with a common goal in a shared environment we use a layered hybrid system [2] containing a state estimation module, an action selection unit, and a multi-robot navigation system. The hybrid architecture works as follows (see fig. 1).

The robot receives sensor data from its camera and some key features from its team mates. Out of that information a model of its environment is built. Thereby principally local information is used while the other robots' key features (if available) are used for evidence. Additionally the robot constructs a model of the environment from each other robot's local view by primarily relying on the key features of that robot. If those features are not available due to communication problems only local information is used.

This technique enables the action selection unit not only to choose an action from a local point of view but to put the robot in another robot's situation and thereby to consider the choice of that robot. This consistency allows the robot to behave cooperative and avoids robots interfering one another. Action selection is done by a simple utility function which is based on a priority list of actions. It is supported by a sophisticated neural projection system which estimates the time need for a requested change in state. Actions are assigned at a frequency of 10Hz and may change depending on the world state even if the respective task was not completed. There is no explicit synchronization between the robots but each robot deciding every 100ms allows only very short times of double assignments.

Each action can be mapped to a certain target state for the robot. All of those target states (consisting of position, orientation etc.) are given to the multi-robot navigation system whose task it is to compute a low level control command leading the robot towards its target state considering obstacle avoidance and motion dynamics.

A Cooperative Action Selection Algorithm. Action selection from a set of actions $\mathcal{A}$ in the context of a multi robot system $\mathcal{R}$ with a common goal means to find a mapping $\mathcal{S} : \mathcal{R} \rightarrow \mathcal{A}$ which assigns each robot of the team $r_i \in \mathcal{R}$ a different action (task) $a_j \in A$ in a way that the common goal can be performed with minimal costs ($c(r_i, a_j)$ is the cost for robot r_i performing action a_j).

$$min \sum_{i=1}^{|\mathcal{R}|} c(r_i, \mathcal{S}(r_i)) \qquad \mathcal{S}(r_i) \not= \mathcal{S}(r_j) \Leftrightarrow i \not= j \tag{1}$$

Since this combinatorial problem is NP-hard usually heuristics are applied for minimizing the cost function. Assuming there are only four robots and four actions given this problem can be solved exactly with the amount of comparing all 24 options.

All the above considerations are based on the assumption that each robot can perform all actions and most notably that the performance of all actions is equally important. But in mine sweeping disabling a mine directly near a civil institution is prior and in robot soccer shooting the ball is more important than any other action in an offensive situation. Furthermore not all robots can execute all actions: A soccer robot without ball cannot shoot the ball and a robot that is stuck cannot move at all. Recapitulatory this means for certain applications the use of a priority list as well as a feasibility list of actions is necessary. Hence

we define a feasibility function

$$\mathcal{F} : \mathcal{R} \times \mathcal{A} \rightarrow \{true,false\} \tag{2}$$

mapping a robot and an action to a boolean value (estimating the probability of success $\mathcal{F}$ can map to $[0, 1]$ too [7]. In this case the following algorithm has to be modified and thresholds are used). In the following algorithm $\mathcal{A}$ ($\hat{\mathcal{R}}$) denotes the priority list of actions (the set of idle robots). This algorithm takes into account that the actions are not equally important and not necessarily feasible for all robots.

$$\begin{array}{l}
\hat{\mathcal{R}} = \mathcal{R} \\
\textit{for } i = 1 \textit{ to } |\hat{\mathcal{R}}| \\
\quad \mathcal{S}(r_i) = \textit{no_operation} \\
\textit{for } i = 1 \textit{ to } |\mathcal{A}| \\
\quad \textit{if } \hat{\mathcal{R}} \neq \emptyset \wedge \min_{r \in \hat{\mathcal{R}}}(c(r, a_i)) < \infty \\
\quad\quad r_j = \arg\min_{r \in \hat{\mathcal{R}}}(c(r, a_i)) \\
\quad\quad \mathcal{S}(r_j) = a_i \\
\quad\quad \hat{\mathcal{R}} = \hat{\mathcal{R}} \setminus r_j
\end{array} \tag{3}$$

The algorithm initializes all robots with the action *no_operation* and then loops for all actions in the order of priority.

$$priority(a_i) > priority(a_j) \Leftrightarrow i < j \tag{4}$$

If there is an idle robot left which can perform the current action the robot with minimal cost to perform this action is chosen to do so. The costs $c(r, a)$ are set to infinity if a robot r cannot perform the action a and to a predicted value $\mathcal{P}(r, a)$ otherwise. The prediction $\mathcal{P}$ depends on the application dependent criterion of optimization.

$$c(r, a) = \begin{cases} \infty & \textit{if} \quad \mathcal{F}(r, a) = \textit{false} \\ \mathcal{P}(r, a) & \textit{otherwise} \end{cases} \tag{5}$$

Using the above algorithm each robot employs $\mathcal{P}$ not only for predicting its own cost but for comparing it with the values computed for its team mates. Relying on identical software and the same local environment data the action selection of this algorithm is unique and consistent. By predicting the cost for robot r_i performing a less important action a_j $\mathcal{P}$ can already rely on the knowledge which robot is to perform a more important action and in case there might be any interference $\mathcal{P}(r_i, a_j)$ will increase. This mechanism forces cooperative behavior and informs all robots on the action selection of their team mates. The major advantages of this approach so far are

- A robot can put itself into the situation of a team mate and thereby will behave cooperative by considering the team mates' decisions.
- All sequences of chosen actions are locally known and toggling situations in action selection can easily be detected and avoided.

- The algorithm is resistant against a crash of a single robot which will be detected (if not moving *and* not communicating for a certain time). Likewise a new robot can easily be integrated in the system. In systems using a high-accuracy state estimation there is even no communication necessary.
- The concept works fine with homogeneous and heterogenous robots. For heterogenous robots $\mathcal{P}$ must be implemented for each different robot.
- The laborious computation of a global map of the environment is not necessary. There is no loss of time for broadcasting a global map.

3 Cooperative Action Selection

3.1 State Estimation

Since action selection strongly depends on a robots view of its environment we first describe how a robot estimates its state.

Probabilistic State Estimation. We employ a state estimation module for individual autonomous robots [25] that enables a team of robots to estimate their joint positions in a known environment (such as a soccer field or an office building) and track the positions of autonomously moving objects. The state estimation modules of different robots cooperate to increase the accuracy and reliability of the estimation process. In particular, the cooperation between the robots enables them to track temporarily occluded objects and to faster recover their position after they have lost track of it.

The state estimation module of a single robot is decomposed into subcomponents for self-localization [13] and for tracking different kinds of objects. This decomposition reduces the overall complexity of the state estimation process and enables the robots to exploit the structures and assumptions underlying the different subtasks of the complete estimation task. Accuracy and reliability is further increased through the cooperation of these subcomponents. In this cooperation the estimated state of one subcomponent is used as evidence by the other subcomponents.

Considering further Physical Properties for State Estimation. So far physical properties like inertia and acceleration of robots are not considered for state estimation. All robots are dealing more or less with a dead time which means that at the moment a robot is performing visual localization it has already sent control commands (e.g. translational and rotational velocity) for further movements. These sent commands can be used to predict the robot's state a little time ahead. This requires a mapping

$$\Delta : \zeta_c \times \xi \mapsto \zeta_{succ} \tag{6}$$

from a current state ζ_c and a sent command ξ to the successor state ζ_{succ}. Δ is learned from experience, that is recorded data from real robot runs. We

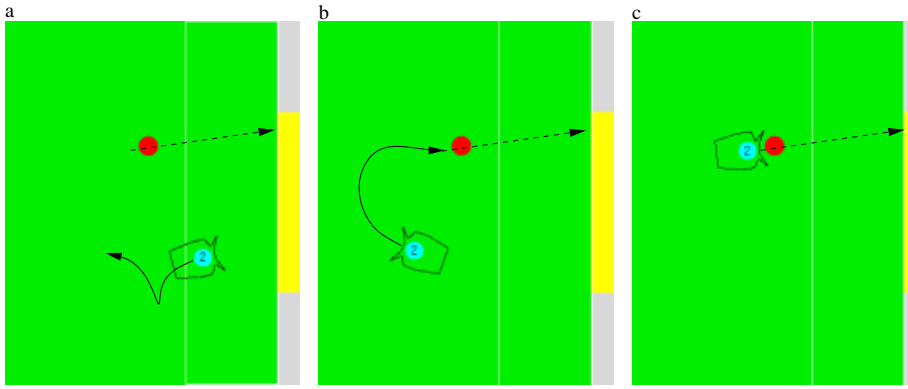


Fig. 2. A training scenario in the multi-robot simulation environment (M-ROSE): To acquire training patterns for the neural projector $\mathcal{P}$ a robot is set to a randomly defined start state ζ_s (position of the robot in subfigure a) and has to drive to a randomly defined target state ζ_t indicated by the dashed arrow. The direction and length of this arrow indicate the target state's orientation and velocity. The time the robot needs to reach its target state (subfigure b and c) is taken to complete the training pattern $\langle\langle\zeta_s, \zeta_t\rangle, time\rangle$

propose to learn this mapping using a simple multi layer neural network and supervised learning with the RPROP algorithm [22]. Depending on the number of command cycles equating with the dead time Δ has to be applied repeatedly to the current state. This enables a robot to perform action selection and path planning considering its future state which is predetermined anyway.

In many mobile robot applications robots can get stuck or blocked by other objects. It is very useful to detect such a situation because it may strongly change the process of action selection not only of the affected robot but of all team mates too. To detect such a situation we employ Δ once again. Recording the low level commands ξ sent to the robot we can predict the robot's changes in state. If they differ significantly from the measured changes in state we assume the robot is not able to move correctly. This information is posted among the robots and will be considered in action selection. If communication fails and a robot does not move a certain time it will be considered dead anyway. Moving again it will be considered alive again.

3.2 A Neural Projection System to Estimate the Time Need for Changes in State in Robotics

As mentioned in section 2 the costs $c(r_i, a_j)$ of a robot r_i performing an action a_j depend on the specific application. In traveling it may be the way, in mine sweeping the time use per mine and in robot soccer the time to score or prevent a goal.

As written above we decompose a goal into several tasks which can be performed by an action each. In our case the costs $c(r_i, a_j)$ of a robot r_i can be

described as the time to complete an action a_j. Assuming that to complete an action means to reach a certain target state we try to predict the time a robot needs to reach that target state. This will give us an information how suitable it is that a robot performs a proposed task. That means we have to estimate the time need of a robot to complete an action considering the following:

- Knowledge of the dynamic behavior of the robot itself must be acquired.
- Actions of team mates must be taken into account.
- Other dynamic objects must be regarded as moving obstacles.

Due to the physical complexity of this problem it seems impossible to estimate the amount of time without learning algorithms. But learning the projection

$$\mathcal{P} : \mathcal{R} \times \mathcal{A} \rightarrow time \tag{7}$$

with data from real robot runs would require an impractical huge amount of time for data acquisition. Less expensive is to construct a robot simulator which mimics the physical behavior of the robots. Our multi-robot simulation environment (M-ROSE,[4]) is based on the state change function Δ (equation 6). Not only the development of a low level controller but the implementation of an action selection unit are substantially simplified by this. Once an accurate level of simulation is achieved one can obtain unlimited training data for learning from such a multi-robot simulator.

Learning a Neural Projection. Neural Networks have been shown to be an accurate means for the prediction of run-times (see Smith et al.[28] for example). Hence we choose neural learning to obtain $\mathcal{P}$. We apply multi layer networks and the back-propagation derivative RPROP [22] because of its adaptive step-size computation in gradient descent. The data we use to train the neural projector $\mathcal{P}$ is completely obtained from the also learned simulator in minimal time. The training patterns are of the form $\langle\langle\zeta_s, \zeta_t\rangle, time\rangle$ where ζ_s is the randomly chosen start state of a robot and ζ_t its also randomly chosen target state in the simulation. We get the necessary value of *time* by simulating a robot driving from ζ_s to ζ_t (see fig. 2). $\mathcal{P}$ was trained with around 300.000 patterns using a network with input, output, and two hidden layers. At learning time there are no other objects or team mates taken into consideration. Using validation patterns for early stopping [24] the trained network achieved an average error of 0.13 seconds per prediction on a test set not used for learning. Due to the inherent indeterministic robot behavior and noise this is an acceptable result.

Taking the Actions of Team Mates into Account. Using the proposed algorithm (3) to select a robot to execute an action a_j we can consider the behavior of all robots executing actions a_i under the condition that $i < j$ (a_i prior to a_j). Thus we can use this knowledge to compute $\mathcal{P}(r, a_j)$ for any robot $r \in \hat{\mathcal{R}}$. Since we know all start and target states of all robots executing actions a_k ($k \leq j$) a set of start and target states as well as the priority of each target

state (action) is given. A multi-robot navigation system receives this data and computes the paths for all concerned robots including r. The multi-robot navigation system (described more in detail in [5,6]) consists of three components: an artificial neural network controller, a library of software tools for planning and plan merging, and a decision module that selects the appropriate planning and execution methods in a situation-specific way. The system has learned predictive models for the performance of different navigation planning methods by experimentation. The decision module uses the learned predictive models to select the most promising planning methods for the given navigation task. If the multi-robot navigation system proposes to apply a path planning method for r to get from its start state ζ_s to its target state ζ_t and the first intermediate state on the computed path is ζ_i the time need is set to

$$\mathcal{P}(\zeta_s, \zeta_t) = \mathcal{P}(\zeta_s, \zeta_i) + \mathcal{P}(\zeta_i, \zeta_t) \tag{8}$$

This equation may be used recursively for the computation of $\mathcal{P}(\zeta_i, \zeta_t)$ if ζ_i is not the last intermediate state on the computed path.

Considering Moving Obstacles. The multi-robot navigation system considers moving obstacles as well. The objective of the navigation system is to achieve a state where each robot is at its target state as fast as possible. A set of representative features considering all obstacles is computed to characterize the navigation task. These features are then tested by a learned decision tree (see [5,6] for more details) that chooses the most promising single robot path planning and plan merging method provided by a toolbox of algorithms. The decision tree was trained using moving obstacles with random direction. The toolbox extracts sequences of target states from the repaired plans and sends those sequences to the neural network controllers of the respective robots.

4 Empirical Investigations

The algorithm described in section 2 has been implemented and extensively tested in simulation and real robot environment. Being highly reliant on cooperation soccer playing robots are a suitable appliance for the approach described above. We observed (a) the behavior of a team of 11 autonomous soccer robots in the RoboCup simulation league environment and (b) the behavior of 4 real robots belonging to the RoboCup MidSize league.

4.1 Simulation Experiments

In the RoboCup simulation environment a soccer team consists of 11 autonomous software agents communicating with a server program [15,20]. The agents receive sensory data and send low level control commands. Using the *Karlsruhe Brainstormers* agent of RoboCup 1999 [21] as a basis for our experiments each robot can avail itself of a set of actions

$\mathcal{A} = \{shoot2goal, pass2player, dribble, receive\_pass, go2ball, offer4pass, gohome\}$

which are mapped to low level commands by the agent. The feasibility function $\mathcal{F}$ is instantiated by hand-coded functions based on situation dependent features. Action selection is done at the frequency of 10Hz. There is no direct communication between the agents but a limited communication via the soccer server simulation program.

We played several games and recorded the locally selected actions of all agents and measured how many times the agent chosen to receive a pass by the agent playing the pass was identic with the agent planning to receive a pass in relation to all passes played. At this we rated a match within a temporal difference of 0.1 seconds (1 simulation cycle) positive. This quotient gives us a measure of cooperation since pass play is a paradigm of cooperation. The experiments were performed with working communication and temporary disordered communication. In the following table one can see the above explained quotient with working communication and with a communication breakdown every 30, 60, and 120 seconds which lasts t_{bd} seconds with t_{bd} uniformly distributed over $[0, 30]$ seconds. Furthermore we played games without any communication. Each result is based on 10 games respectively.

Communication	Matches (pass player / receiver)	Average goals per game
full	82.1%	8.1
breakdown each 120 s	76.2%	7.4
breakdown each 60 s	70.1%	6.5
breakdown each 30 s	66.1%	6.1
no communication	63.9%	5.7

One can see that an increasing frequency of communication breakdown accompanies with a decreasing number of goals scored per game. Moreover less communication means less successful planning of pass play and therewith less coordination. Nevertheless a team using no communication is able to score around 70% of the goals a team with full communication scores. Additionally 77.8% of the matching quotient of a team using full communication is reached by a team without any communication. These results document that the chosen approach for action selection and coordination is robust and can deal very well with temporary failures.

Double Pass Play. Without ever explicitly learning to play a double pass and no special double-pass-plan specified not seldom double pass play was observed. Analyzing some cases of double pass play we found a reasonable pattern to explain this effect (see fig. 3). Robot number 3 holds the ball and decides to pass to player number 2. Meanwhile player number 2 puts itself in the situation of player number 3 and recognizes that it has to receive a pass:

$$\mathcal{S}(r_3) = pass2player_2 \qquad \mathcal{S}(r_2) = receive\_pass_3$$

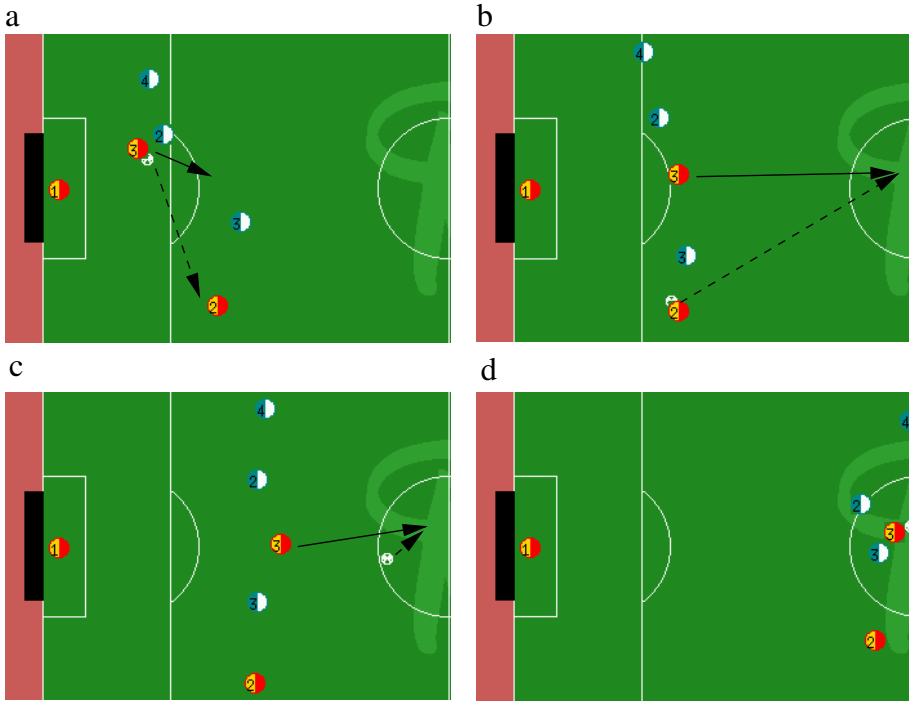


Fig. 3. A double pass scenario in the RoboCup soccer server environment. Player 3 chooses to play a pass to player 2 while at the same time player 2 awaits the ball (subfigure a). After player 3 gets rid of the ball he starts to offer itself for a pass play. As player 2 gets the ball player 3 computes that player 2 will pass the ball again (subfigure b). Finally player 3 gets back the ball (subfigure c and d)

Immediately after having played the ball player 3 performs $\mathcal{S}(r_3) = \mathit{offer4pass}$ to offer itself for a receipt of a pass (fig. 3a). As player 2 receives the ball it chooses to play a pass to player 3. Meanwhile player 3 puts itself in the situation of player number 2 and recognizes that it has to receive a pass (fig. 3b):

$$\mathcal{S}(r_2) = \mathit{pass2player}_3 \qquad \mathcal{S}(r_3) = \mathit{receive\_pass}_2$$

Further player 3 performs $\mathcal{S}(r_3) = \mathit{receive\_pass}$ while player 2 performs $\mathcal{S}(r_2) = \mathit{offer4pass}$ after playing the pass back to player 3 (fig. 3c). Receiving the ball again player 3 is to choose from $\{\mathit{shoot2goal}, \mathit{pass2player}, \mathit{dribble}\}$ again (fig. 3d).

4.2 Real Robot Experiments

To evaluate our approach in a real robot environment we choose the RoboCup MidSize league. Two teams of 4 players compete on a field of about 9 meters in length and 5 meters in width. Compared with the RoboCup simulation league there are some new challenging problems: The sensory data is not provided by a server program but must be acquired by the robot itself. Furthermore most

robots are not able to receive a pass from any direction (like in simulation league) and path planning is more important on a comparably small field (the field in simulation league is 105 meters long). For our experiments we use the Agilo RoboCuppers team [3] as a basis. The available list of actions looks as follows:

$$\mathcal{A} = \{shoot2goal, dribble, clear\_ball, go2ball, gohome, get\_unstuck\}$$

To demonstrate the coordination of the team we measure the number of robots performing *go2ball* at the same time. Further we observe how long the same robot performs *go2ball* without being interrupted by another robot. The data was acquired from five real robot games against different opponent teams at the international robot soccer world cup 2001. The following table depicts in how much percent of the whole time played none, one, and more than one robot performed *go2ball*. The frequency of action selection is around 10Hz.

#robots performing *go2ball* at the same time	quota in relation to the whole time played
0	00.34%
1	98.64%
> 1	01.02%

The average time one robot performs *go2ball* or handles the ball without being interrupted by a decision of a team mate is 3.35 seconds. In only 0.25% of the time a robot that is stuck is determined to go for the ball by the other robots.

These results show that, in the context of robot soccer, coordination is warranted to a great extent. There are hardly ever situations where no robot or more than one robot approaches the ball at a time and the situations in which it is not clear which robot is to go for the ball are just a few.

Solving Situations with Stuck Robots. In a highly dynamic environment like robot soccer not seldom a robot gets stuck due to another robot blocking it. The following example shows how such incidents were handled by the robots in our experiments (see fig. 4). Robot number 2 is supposed to be the fastest to get the ball and therefore approaches the ball (fig. 4a): $\mathcal{S}(r_2) = go2ball$. Near the ball robot 2 collides with an opponent robot. Robot 2 is in a deadlock situation and cannot move forward anymore. The only action feasible to execute remains *get_unstuck*. Thus robot 3 approaches the ball now (fig. 4b):

$$\mathcal{F}(r_2, a) = 0 \; \forall a \in \mathcal{A} \setminus get\_unstuck \quad \mathcal{S}(r_2) = get\_unstuck \quad \mathcal{S}(r_3) = go2ball$$

Having reached the ball robot 3 dribbles towards the opponent goal while robot 2 is moving backwards (fig. 4c):

$$\mathcal{F}(r_2, a) = 0 \; \forall a \in \mathcal{A} \setminus get\_unstuck \quad \mathcal{S}(r_2) = get\_unstuck \quad \mathcal{S}(r_3) = dribble$$

Further on robot 2 is no more stuck and robot 3 is still dribbling:

$$\mathcal{F}(r_2, go2ball) = 1 \qquad \mathcal{S}(r_2) = gohome \qquad \mathcal{S}(r_3) = dribble$$

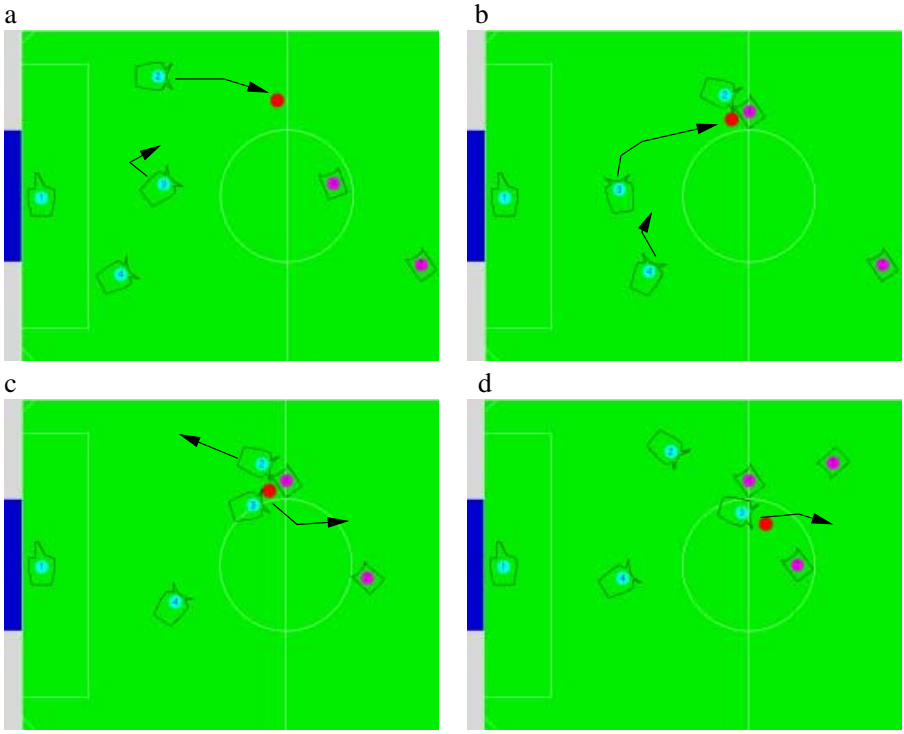


Fig. 4. An example for intelligent cooperation in a real robot soccer environment. Robot 2 approaches the ball (subfigure a) and thereby collides with a robot of the opponent team (subfigure b). As the opponent robot constantly pushes robot 2 is stuck and temporary not regarded by the other robots. Thus robot 3 moves towards the ball while robot 2 tries to get unstuck (subfigure c). Finally robot 3 dribbles towards the opponent goal while robot 2 is staying back in its own half (subfigure d)

5 Conclusions

In this paper we propose an autonomous approach to collaborative action selection for multi-robot environments. Action selection directly depends on the model of the environment. Our state estimation process of a single robot is mainly based on local sensor data while information from other robots is used for evidence only. Decision making is supported by the ability of a robot to put itself in its team mates' situation. This mechanism used by humans and apes allows cooperative behavior among the robots. Moreover employing a neural projection system each robot can estimate the time need for itself or other robots to reach a certain state. Our technique has been implemented and tested in the highly dynamic and cooperation-dependent RoboCup environment both in simulation and in real robot runs. The results show that the proposed approach is reliable and, to a high extent, fault tolerant even if there is no communication between the robots but a reliable localization.

Compared to most previous methods our concept is neither leader-following nor dependent on a shared global map of the environment. Each robot decides completely autonomous supported by neural prediction. In action selection we consider the feasibility of an action as well as its costs and its priority. The main advantages of our method are extensibility, the applicability to heterogenous robots, stability in situations of failures of single robots, the consistency in action assignment, and that the laborious computation of a shared global map is not necessary.

Multi-robot action selection in common is assumed to be a combinatorial problem that requires an exponential computational amount to be solved exactly. However we have to consider feasibility and priority of actions as well. To improve performance in the aspect of time sophisticated heuristics are required. We believe that a reasonable set of executable actions as well as the optimization in performance of every single action will substantially support the effectiveness of our algorithm. These extensions are subject of our future investigations as well as the integration of long term plans and the extension to other applications.

Acknowledgments

We would like to thank our students (Maximilian Fischer and Andreas Hofhauser) for their contributions to our RoboCup team. Further thanks go to Artur Merke and Martin Riedmiller (members of the RoboCup Simulation Team *Brainstormers*) for supporting the simulation experiments and providing the *n++* neural network library. Finally we would like to thank the staff and the organizers of the *Schloss Dagstuhl Seminar* for the event and the *RoboCup Federation* and the *DFG* for supporting our work.

References

1. R. Alur, J. Esposito, M. Kim, V. Kumar, and I. Lee: *Formal modeling and analysis of hybrid systems: A case study in multirobot coordination.* FM'99: Proceedings of the World Congress on Formal Methods, LNCS 1708, pp. 212–232, Springer, 1999.
2. R.C. Arkin: *Towards the Unification of Navigational Planning and Reactive Control.* AAAI Spring Symposium on Robot Navigation, pp. 1-5, 1989.
3. M. Beetz, S. Buck, R. Hanek, T. Schmitt, and B. Radig: *The Agilo Autonomous Robot Soccer Team: Computational Principles, Experiences, and Perspectives.* International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS) 2002.
4. S. Buck, M. Beetz, and T. Schmitt: *M-ROSE: A Multi-Robot Simulation Environment for Learning Cooperative Behavior.* In H. Asama, T. Arai, T. Fukuda, and T. Hasegawa (eds.): Distributed Autonomous Robotic Systems 5, 2002, Springer Verlag.
5. S. Buck, U. Weber, M. Beetz, and T. Schmitt: *Multi-Robot Path Planning for Dynamic Environments: A Case Study.* Proceedings of the IEEE International Conference on Intelligent Robots and Systems, 2001.

6. S. Buck, M. Beetz, and T. Schmitt: *Planning and Executing Joint Navigation Tasks in Autonomous Robot Soccer*. 5th International Workshop on RoboCup, Lecture Notes in Artificial Intelligence, Springer Verlag, 2001.
7. S. Buck and M. Riedmiller: *Learning Situation Dependent Success Rates Of Actions In A RoboCup Scenario*. Proceedings of the Pacific Rim International Conference on Artificial Intelligence, Lecture Notes in Artificial Intelligence, Springer 2000.
8. W. Burgard, M. Moors, D. Fox, R. Simmons, and S. Thrun: *Collaborative multi-robot localization*. Proceedings of the IEEE International Conference on Robotics and Automation, 2000.
9. Y.U. Cao, A.S. Fukunaga, and A.B. Khang: *Cooperative mobile robotics: Antecedents and directions*. Autonomous Robots, 4, 1997.
10. Q. Chen and J.Y.S. Luh: *Coordination and control of a group of small mobile robots*. Proceedings of the IEEE International Conference on Robotics and Automation, pp. 2315-2320, 1994.
11. G. Dudek, M. Jenkin, E.E. Milios, and D. Wilkes: *A taxonomy for multi-agent robotics*. Autonomous Robots, 3(4), 1996.
12. A. Garland and R. Alterman: *Multiagent Learning through Collective Memory*. AAAI Spring Symposium Series 1996: Adaption, Co-evolution and Learning in Multiagent Systems, 1996.
13. R. Hanek and T. Schmitt: *Vision-Based Localization and Data Fusion in a System of Cooperating Mobile Robots*. Proceedings of the IEEE International Conference on Intelligent Robots and Systems, pages 1199-1204, 2000.
14. N. Jennings: *Controlling cooperative problem solving in industrial multi-agent systems using joint intentions*. Artificial Intelligence, 75, 1995.
15. H. Kitano, M. Tambe, P. Stone, S. Coradesci, H. Matsubara, M. Veloso, I. Noda, E. Osawa, and M. Asada: *The robocup synthetic agents' challenge*. In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), 1997.
16. R. Kube and H. Zhang: *Collective Robotics: From Social Insects to Robots*. Adaptive Behaviour, Vol. 2, No. 2, pp. 189-218, 1994.
17. H.J. Levesque, P.R. Cohen, and J. Nunes: *On acting together*. In Proceedings of the National Conference on Artificial Intelligence, AAAI press, 1990.
18. M.J. Mataric: *Coordination and Learning in Multi-robot Systems*. IEEE Intelligent Systems, Mar/Apr 1998, 6-8.
19. M.J. Mataric: *Using Communication to Reduce Locality in Distributed Multi-Agent Learning*. Journal of Experimental and Theoretical Artificial Intelligence, special issue on Learning in DAI Systems, Gerhard Weiss, ed., 10(3), Jul-Sep, 357-369, 1998.
20. I. Noda, H. Matsubara, K. Hiraki, and I. Frank: *Soccer Server: A Tool for Research on Multiagent Systems*. Applied Artificial Intelligence, 12, 2–3, pp.233–250, 1998.
21. M. Riedmiller, S. Buck, A. Merke, R. Ehrmann, O. Thate, S. Dilger, A. Sinner, A. Hofmann, and L. Frommberger: *Karlsruhe Brainstormers - Design Principles*. In M. Veloso, E. Pagello, H. Kitano, editors, RoboCup-99: Robot Soccer World Cup III, Lecture Notes in Artificial Intelligence, pp 588-591, Springer Verlag, 1999.
22. M. Riedmiller and H. Braun: *A direct adaptive method for faster backpropagation learning: the Rprop algorithm*, Proceedings of the ICNN, San Francisco, 1993.
23. A. Saffiotti, N. B. Zumel, and E. H. Ruspini: *Multi-robot Team Coordination using Desirabilities*. Proceedings of the Sixth International Conference on Intelligent Autonomous Systems, 2000.
24. W.S. Sarle: *Stopped training and other remedies for overfitting*. In Proceedings of the 27th Symposium on Interface, 1995.

25. T. Schmitt, R. Hanek, S. Buck, and M. Beetz: *Cooperative Probabilistic State Estimation for Vision-based Autonomous Mobile Robots*. Proceedings of the IEEE International Conference on Intelligent Robots and Systems, 2001.
26. S. Sen, S. Mahendra, and J. Hale: *Learning to Coordinate Without Sharing Information*. Proceedings of the Twelfth National Conference on Artificial Intelligence, pp. 426-431, 1994.
27. R.G. Simmons, D. Apfelbaum, W. Burgard, D. Fox, M. Moors, S. Thrun, and H. Younes: *Coordination for multi-robot exploration and mapping*. In Proceedings of the Seventeenth National Conference on Artificial Intelligence, pp. 852-858, 2000.
28. W. Smith, V.E. Taylor, and I. Foster: *Using Run-Time Predictions to Estimate Queue Wait Times and Improve Scheduler Performance*. JSSPP, pp. 202-219, 1999.
29. M. Tambe and W. Zhang: *Towards flexible teamwork in peristent teams*. Proceedings of the International conference on multi-agent systems (ICMAS), 1998.
30. A. Tews and G. Wyeth: *Thinking as One: Coordination of Multiple Robots by Shared Representations*. Proceedings of the IEEE International Conference on Intelligent Robots and Systems, vol. 2, pp. 1391-1396, 2000.
31. A. Tews and G. Wyeth: *Multi-robot Coordination in the Robot Soccer Environment*. Proceedings of the Australian Conference on Robotics and Automation (ACRA '99), March 30 - April 1, Brisbane, pp. 90-95, 1999.
32. B.J. Young, R.W. Beard, J.M. Kelsey: *Coordinated Control of Multiple Robots using Formation Feedback*. IEEE Transactions on Robotics and Automation, In Review.

Collaborative Exploration of Unknown Environments with Teams of Mobile Robots

Wolfram Burgard[1], Mark Moors[2], and Frank Schneider[3]

[1] Department of Computer Science, University of Freiburg, 79110 Freiburg, Germany
[2] Department of Computer Science, University of Bonn, 53117 Bonn, Germany
[3] Research Establishment for Applied Science, 53343 Wachtberg, Germany

Abstract In this paper we consider the problem of exploring an unknown environment by a team of robots. As in single-robot exploration the goal is to minimize the overall exploration time. The key problem to be solved in the context of multiple robots is to choose appropriate target points for the individual robots so that they simultaneously explore different regions of the environment. We present an approach for the coordination of multiple robots which, in contrast to previous approaches, simultaneously takes into account the cost of reaching a target point and its utility. The utility of a target point is given by the size of the unexplored area that a robot can cover with its sensors upon reaching that location. Whenever a target point is assigned to a specific robot, the utility of the unexplored area visible from this target position is reduced for the other robots. This way, a team of multiple robots assigns different target points to the individual robots. The technique has been implemented and tested extensively in real-world experiments and simulation runs. The results given in this paper demonstrate that our coordination technique significantly reduces the exploration time compared to previous approaches.

1 Introduction

The problem of exploring an environment belongs to the fundamental problems in mobile robotics. There are several applications like planetary exploration [3], reconnaissance [26], rescue, mowing [28], or cleaning [19, 48] in which the complete coverage of a terrain belongs to the inherent goals of a robotic mission.

In this paper, we consider the problem of exploring unknown environments with teams of mobile robots. The use of multiple robots is often suggested to have several advantages over single robot systems [9, 17]. First, cooperating robots have the potential to accomplish a single task faster than a single robot [25]. Furthermore, using several robots introduces redundancy. Teams of robots therefore can be expected to be more fault-tolerant than only one robot. Another advantage of robot teams is due to merging of overlapping information, which can help compensate for sensor uncertainty. For example, multiple robots have been shown to localize themselves more efficiently, especially when they have different sensor capabilities [21]. However, when robots operate in teams there is the

M. Beetz et al. (Eds.): Plan-Based Control of Robotic Agents, LNAI 2466, pp. 52–70, 2002.

risk of possible interferences between them [20, 22]. For example, if the robots have the same type of active sensors such as ultrasound sensors, the overall performance can be reduced due to cross-talk between the sensors. Furthermore, the more robots are used the longer detours may be necessary in order to avoid collisions with other members of the team.

In this paper we present an algorithm for coordinating a group of robots while they are exploring their environment. Our method, which has originally been presented in [40] and has been integrated into two different systems [8, 47], follows a decision-theoretic approach. Instead of greedily guiding every robot to the closest unexplored area, our algorithm explicitly coordinates the robots. It tries to maximize overall utility by minimizing the potential for overlap in information gain amongst the various robots. Our algorithm simultaneously considers the utility of unexplored areas and the cost for reaching these areas. By trading off the utilities and the cost and by reducing the utilities according to the number of robots that already are heading towards this area, coordination is achieved in a very elegant way. The underlying mapping algorithm, which is described in detail in [52], is an on-line solution to the *simultaneous localization and mapping problem (SLAM)* [10, 15]. In a distributed fashion it computes a consistent representation of the environment explored so far and also determines the positions of the robots given this map.

Our technique has been implemented on teams of heterogeneous robots and has been proven effectively in realistic real-world scenarios. Additionally we have carried out a variety of simulation experiments to explore the properties of our approach and to compare the coordination mechanism to other approaches developed so far. As the experiments demonstrate, our technique significantly reduces the time required to completely cover an unknown environment with a team of robots.

2 Coordinating a Team of Robots During Exploration

The goal of an exploration process is to cover the whole environment in a minimum amount of time. Therefore, it is essential that the robots keep track of which areas of the environment have already been explored. Furthermore, the robots have to construct a global map in order to plan their paths and to coordinate their actions. Throughout this section we assume that at every point in time both, the map of the area explored so far and the positions of the robots in this map are known. The focus of this section lies in the question of how to coordinate the robots in order to efficiently cover the environment. The mapping system will briefly be described Section 3.

Our system uses occupancy grid maps [41, 52] to represent the environment. Each cell of such an occupancy grid map contains a numerical value representing the probability that the corresponding area in the environment is covered by an obstacle. Since the sensors of real robots generally have a limited range and since often parts of the environment are occluded by objects, a map generally contains certain cells whose value is "unknown" since they have never been updated so

far. Throughout this paper, we assume that exploredness is a binary concept and we regard a cell as explored as soon as they have been covered by a sensor beam.

When exploring an unknown environment we are especially interested in "frontier cells" [53]. As a frontier cell we denote each already explored cell that is an immediate neighbor of an unknown, unexplored cell. If we direct a robot to such a cell, we can expect that it gains information about the unexplored area when it arrives at its target location. The fact that a map generally contains several unexplored areas raises the problem that there often are multiple possibilities of directing robots to frontier cells. On the other hand, if multiple robots are involved, we want to avoid that several of them move to the same location. Our system uses a decision-theoretic framework approach to determine appropriate target locations for the individual robots. We simultaneously consider the cost of reaching a frontier cell and the utility of that cell. For each robot, the cost of a cell are proportional to the distance between the robot and that cell. The utility of a frontier cell instead depends on the number of robots that are moving to that cell or to a place close to that cell.

In the following sections we will describe how we compute the cost of reaching a frontier cell for the individual robots, how we determine the utility of a frontier cell and how we choose appropriate assignments of robots to frontier cells.

2.1 Costs

To determine the cost of reaching the current frontier cells, we compute the optimal path from the current position of the robot to all frontier cells based on a deterministic variant of *value iteration*, a popular dynamic programming algorithm [5, 27]. In our approach, the cost for traversing a grid cell $\langle x, y\rangle$ is proportional to its occupancy value $P(occ_{xy})$. The minimum-cost path is computed using the following two steps.

1. **Initialization.** The grid cell that contains the robot location is initialized with 0, all others with ∞:

$$V_{x,y} \longleftarrow \begin{cases} 0, & \text{if } \langle x, y\rangle \text{ is the robot position} \\ \infty, & \text{otherwise} \end{cases}$$

2. **Update Loop.** For grid cells $\langle x, y\rangle$ do:

$$V_{x,y} \longleftarrow \min_{\substack{\Delta x = -1,0,1 \\ \Delta y = -1,0,1}} \left\{ V_{x+\Delta x, y+\Delta y} + \sqrt{\Delta x^2 + \Delta y^2} \cdot P(occ_{x+\Delta x, y+\Delta y}) \right\}$$

This technique updates the value of all grid cells by the value of their best neighbors, plus the cost of moving to this neighbor. Here, cost is equivalent to the probability $P(occc_{x,y})$ that a grid cell $\langle x, y\rangle$ is occupied times the distance to the cell. The update rule is iterated. When the update converges, each value $V_{x,y}$ measures the *cumulative cost* for moving to the corresponding cell. The resulting

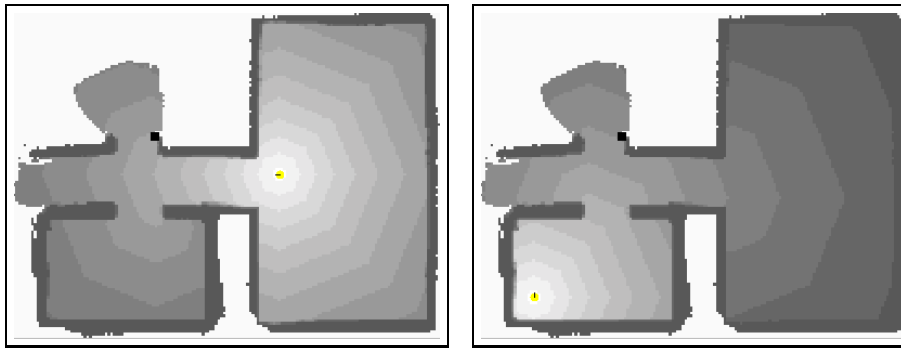


Figure 1. Typical value functions obtained for two different robot positions. The black rectangle indicates the target points in the unknown area with minimum cost

value function V can also be used to efficiently derive the minimum-cost path from the current location of the robot to arbitrary goal positions. This is done by steepest descent in V, starting at the desired goal position.

Figure 1 shows the resulting value functions for two different robot positions. The black rectangle indicates the target point in the unknown area with minimum travel cost. Please note that the same target point is chosen in both situations. Accordingly, if the robots are not coordinated during exploration, they would move to the same position which obviously is not optimal.

Our algorithm differs from standard value iteration in that it regards all actions of the robots as deterministic. This way, the value function can be computed faster than with value iteration. To incorporated the uncertainty of the robots motions into our approach and to benefit from the efficiency of the deterministic variant, we smooth the input maps by a convolution with a Gaussian kernel. This has a similar effect as generally observed when using the non-deterministic approach: It introduces a penalty for traversing narrow passages or staying close to obstacles. Therefore, the robots generally prefer target points in open spaces rather than behind narrow doorways. Please note that the maps depicted in Figure 1 are not smoothed.

2.2 Computing Utilities of Frontier Cells

Estimating the utility of frontier cells is more difficult. In fact, the actual information that can be gathered by moving to a particular location is impossible to predict, since it very much depends on the structure of the corresponding area. However, if there already is a robot that is moving to a particular frontier cell, the utility of that cell can be expected to be lower for other robots. But not only the designated target location has a reduced utility. Since sensors of a robot also cover the terrain around a frontier cell as soon as the robot arrives there, even

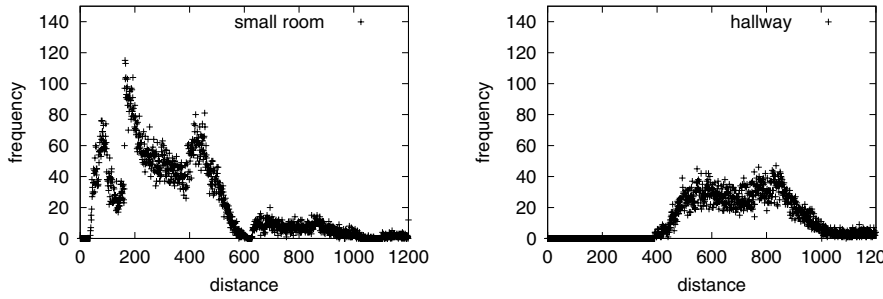


Figure 2. Distance histograms $h(d \mid s)$ obtained in a small room (left) and in a hallway (right)

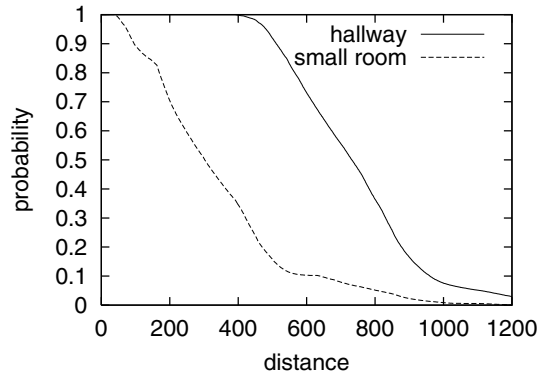


Figure 3. Resulting likelihood $P(d)$ of measuring at least distance d for the histograms depicted in Figure 2

the expected utility of frontier cells in the vicinity of the robot's target point is reduced.

In this section we will present a technique that estimates the expected utility of a frontier cell based on the distance to cells that are assigned to other robots. To adapt the system to the structure of the environment, we permanently and on-line estimate the visibility range of the sensors of all robots. Suppose in the beginning each frontier cell t has the utility U_t which is equal for all frontier cells if no additional information about the usefulness of certain positions in the environment is available. Whenever a target point t' is selected for a robot, we reduce the utility of the adjacent frontier cells in distance d from t' according to the probability $P(d)$ that the robot's sensors will cover cells in distance d.

To compute the quantity $P(d)$ while the robots are exploring the environment we count for a discrete set of distances $d_1, \ldots, d_n$ the number of times $h(d_i)$ the

distance d_i was measured by any of the robots. Based on this histogram we can compute the probability $P(d)$ that a cell in distance d will be covered by a sensor beam:

$$P(d) = \frac{\sum_{d_i \geq d} h(d_i)}{\sum_{d_i} h(d_i)} \tag{1}$$

Thus, any cell t in distance d from the designated target location t' will be covered with probability $P(d)$ when the robot reaches t'. Accordingly, we compute the utility $U(t_n \mid t_1, \ldots, t_{n-1})$ of a frontier cell t_n given that the cells $t_1, \ldots, t_{n-1}$ have already been assigned to the robots $1, \ldots, n-1$ as

$$U(t_n \mid t_1, \ldots, t_{n-1}) = U_{t_n} - \sum_{i=1}^{n-1} P(||t_n - t_i||) \tag{2}$$

According to Equation 2, the more robots move to a location from where t_n is likely to be visible, the lower is the utility of t_n.

The advantage of this approach is that it automatically adapts itself according to the free space in the environment. For example, in an area with wide open spaces, such as a hallway, the robots are expected to sense a higher number of long readings than in narrow areas or small rooms. As an example consider the two different histograms depicted in Figure 2. Here a team of robots started in a large open hallway (left image) and in a typical office room (right image). Obviously the robots measure shorter readings in rooms than in a hallway. Correspondingly, the probability of measuring at least $4m$ is almost one in the hallway whereas it is comparably small in a room (see Figure 3). Please note that we also take into account whether there is an obstacle between two frontier cells t and t'. This is achieved using a ray-tracing on the grid map. If there is an obstacle in between, we set $P(||t - t'||)$ to zero.

2.3 Target Point Selection

To compute appropriate target points for the individual robots we need to consider for each robot (1) the cost of moving to a location and (2) the utility of that location. In particular, for each robot i we trade-off the cost V_t^i to move to the location t and the utility U_t of t.

To determine appropriate target points for all robots, we use an iterative approach together with a greedy strategy. In each round we compute that tuple of a robot i and a target point t, which has the best overall evaluation $U_t - \beta \cdot V_t^i$. Here $\beta \geq 0$ determines the relative importance of utility versus cost. In the decision-theoretic context the choice of β usually depends on the application. In our system, β generally was set to 1. We then recompute the utilities of all frontier cells given the new and all previous assignments according to Equation 2. This results in the algorithm shown in Table 1.

Figure 4 illustrates the effect of our coordination technique. Whereas uncoordinated robots would choose the same target position (see Figure 1), the coordinated robots select different frontier cells as the next exploration targets.

Table 1. The Target Point Selection Algorithm with Greedy Assignment

1. Determine the set of frontier cells
2. Compute for each robot i the cost V_t^i for reaching each frontier cell t
3. Set the utility U_t of all frontier cells to 1
4. While there is one robot left without a target point
   (a) Determine a robot i and a frontier cell t which satisfy

$$(i,t) = \underset{(i',t')}{\operatorname{argmax}} \left(U_{t'} - \beta \cdot V_{t'}^{i'} \right) \tag{3}$$

   (b) Reduce the utility of each target point t' in the visibility area according to

$$U_{t'} \leftarrow U_{t'} - P(||t - t'||) \tag{4}$$

Please note that in Step 4.a the assignment is computed quite efficiently. The complexity is $O(n^2 m)$ where n is the number of robots and m is the number of frontier cells. In principle, one could also try to find the optimal assignment instead. This however, introduces a problem that one has to iterate over all possible assignments of n robots to m frontier cells. In the experimental section we will also consider an approach that optimizes the assignment in such a way. Whereas this method has been found to yield slightly better results in certain environments it is much more time-consuming.

3 Collaborative Mapping with Teams of Mobile Robots

To explore their environment and to coordinated their actions, the robots need a detailed map of the environment. Furthermore, the robots must be able to build maps online, while they are in motion. The online characteristic is especially important in the context of the exploration task, since mapping is constantly interleaved with decision making as to where to move next.

To map an environment, a robot has to cope with two types of sensor noise: Noise in perception (e.g., range measurements), and noise in odometry (e.g., wheel encoders). Because of the latter, the problem of mapping creates an inherent localization problem, which is the problem of determining the location of a robot relative to its own map. The mobile robot mapping problem is therefore often referred to as the *concurrent mapping and localization problem (CML)* [36], or as the *simultaneous localization and mapping problem (SLAM)* [10, 15].

Our system applies the statistical framework presented in detail in [52] to compute consistent maps while the robots are exploring the environment. Each robot simultaneously performs two tasks: It determines a maximum likelihood estimate for its own position and a maximum likelihood estimate for the map

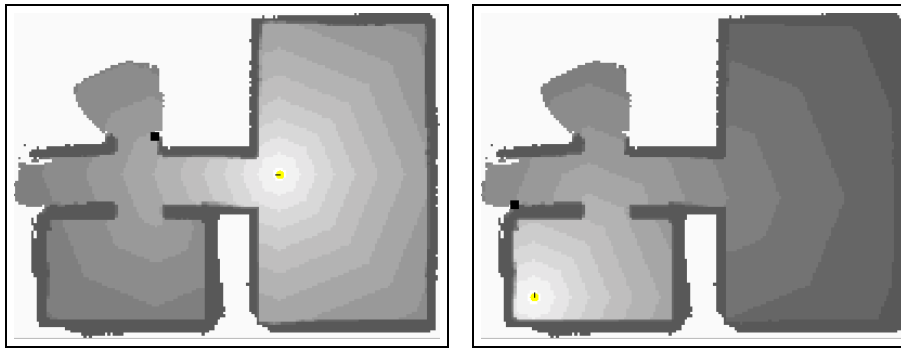


Figure 4. Target positions obtained using the coordination approach. In this case the target point for the second robot is to the left in the corridor

(location of surrounding objects). To recover from possible localization errors, each robot maintains a posterior density characterizing its "true" location ([52]). The whole process is carried out in a distributed fashion. A central module receives the local maps and combines them into a single, global map which then is broadcasted to all robots. The current version of the system relies on the following two assumptions:

1. The robots must begin their operation in nearby locations, so that their range scans show substantial overlap.
2. The software must be told the approximate relative initial pose of the robots. Thereby errors up to 50 cm and 20 degrees in orientation are admissible.

4 Experimental Results

The approach described has been implemented and extensively tested on real robots and in different environments. Additionally to the experiments carried out using real robots we performed a series of simulation experiments to get a quantitative assessment of the improvements of our approach over previous techniques.

4.1 Exploration with a Team of Mobile Robots

The first experiment is designed to demonstrate the capability of our approach to efficiently cover an unknown environment with a team of mobile robots. To evaluate our approach we installed three robots (two Pioneer I and one RWI B21) in an empty office environment. Figure 5 shows the map of the environment. The size of this environment is $18 \times 14m$. Also shown are the paths of the robots which started in the upper left office. As can be seen from the figure, the robots were effectively distributed over the environment. This demonstrates that our

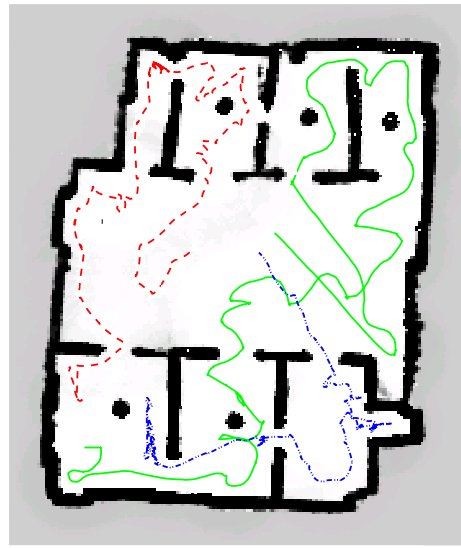


Figure 5. Coordinated exploration by a team of three robots

approach can effectively guide a team of mobile robots to collaboratively explore an unknown environment.

4.2 Comparison between Greedy and Coordinated Exploration

The experiment described in this section is designed to illustrate the advantage of our coordination technique over an approach in which the robots share a map but in which there is no arbitration about target locations so that each robot approaches the closest frontier cell. Typical techniques belonging to this class are described in [53, 49]. For this experiment we used two different robots: An RWI B21 robot equipped with two laser-range scanners and a Pioneer I robot equipped with a single laser scanner. The size of the environment to be explored in this experiment was $14 \times 8m$ and the range of the laser sensors was limited to $5m$.

Figure 6 shows the typical behavior of the two robots when they explore their environment without coordination, i.e. when each robot moves to the closest unexplored location. The white arrows indicate the positions and directions of the two robots. Since the cost for moving through the narrow doorway in the upper left room are higher than the cost for reaching a target point in the corridor, both robots decide first to explore the corridor. After reaching the end of the corridor one robot enters the upper right room. At that point the other robot assigns the highest utility to the upper left room and therefore turns back. Before this robot reaches the upper left room, the B21 platform

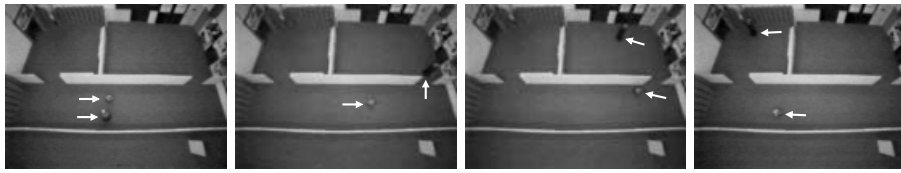


Figure 6. Uncoordinated exploration with two robots

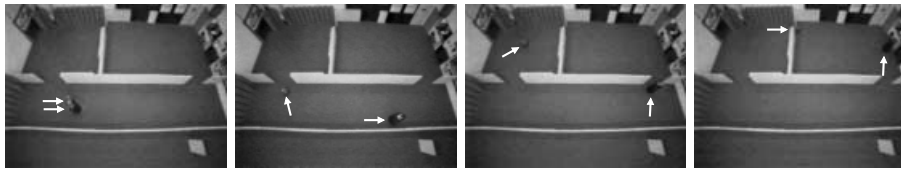


Figure 7. Coordinated exploration by two robots

has already entered it and has completed the exploration mission. As a result, the B21 system explores the whole environment on its own and the Pioneer I robot does not contribute anything. The overall time needed to complete the exploration was 49 seconds in this case.

If, however, both robots are coordinated they perform much better (see Figure 7). As in the previous example, the B21 system moves to the end of the corridor. Since the utilities of the frontier cells in the corridor are reduced, the Pioneer I platform directly enters the upper left room. As soon as both robots have entered the rooms, the exploration mission is completed. This run lasted 35 seconds.

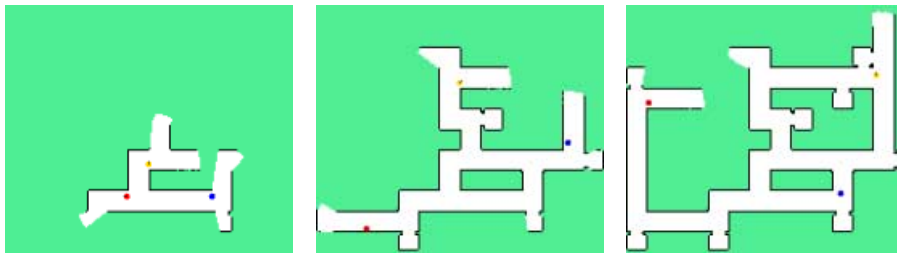


Figure 8. Simulated exploration with three robots

4.3 Simulation Experiments

The previous experiments demonstrate that our approach can effectively guide robots to collaboratively explore an unknown environment. To get a more quantitative assessment we performed a series of simulation experiments in using

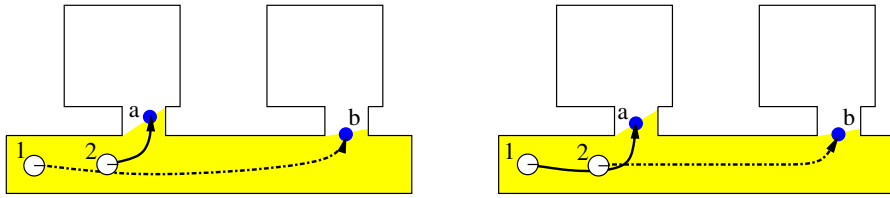


Figure 9. The trajectories depicted in the left image that result from algorithm 1 are sub-optimal. If robot 1 moves to point a and robot 2 moves to the location b as illustrated in the right figure, the time needed to finish the exploration task is reduced, since the maximum time needed to reach the rooms is lower

different environments. For this purpose, we developed a simulation system, that allows us to consider the effects of various parameters on the exploration performance. The simulator can handle an arbitrary number of robots. It uses a discretized representation of the state space into equally sized cells of $15 \cdot 15cm$ and 8 orientations. Additionally, it models interferences between the robots using a randomized strategy. Whenever the robots are close to each other, the system performs the planned movement with a probability of 0.7. Thus, robots that stay close to each other move slower than robots that are isolated.

Throughout these experiments we compared three different strategies. The first approach is the technique used by Yamauchi et al. [53] as well as [49], in which all robots share a joint map and greedily approach the closest unexplored part of the map. The second approach is our coordination algorithm shown in Table 1.

Additionally, we evaluated an alternative approach that seeks to optimize the assignments computed in Step 4 of our algorithm. For example, consider the situation depicted in Figure 9. Here two robots are exploring a corridor with two offices. The already explored area is depicted in grey/yellow. The assignment resulting from an application of our algorithm is depicted in the left image of this figure. Suppose both target points a and b have the same utility. Then in the first round our algorithm assigns robot 2 to a since this assignment has the least cost of all other possible assignments. Accordingly, in the second round, 1 is assigned to b.

If we assume that both robots require the same amount of time to explore a room, this assignment is clearly sub-optimal. A better assignment is shown in the right image of Figure 9. By directing robot 1 to the left room and robot 2 to the right room, the whole team can finish the job earlier, because the time required to reach the rooms is reduced.

As already mentioned above, one approach to overcome this problem is to consider all possible combinations of target points and robots. Again we want to minimize the trade-off between the utility of frontier cells and the distance to be traveled. However, just adding the distances to be traveled by the two robots

Table 2. Target point selection determining th optimal assignment

1. Determine the set of frontier cells
2. Compute for each robot i the cost V_t^i for reaching each frontier cell
3. Determine target locations $t_1, \ldots, t_n$ for the robots $i = 1, \ldots, n$ that maximizes the following evaluation function

$$\sum_{i=1}^{n} U(t_i \mid t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_n) - \beta \cdot (V_{t_i}^i)^2 \tag{6}$$

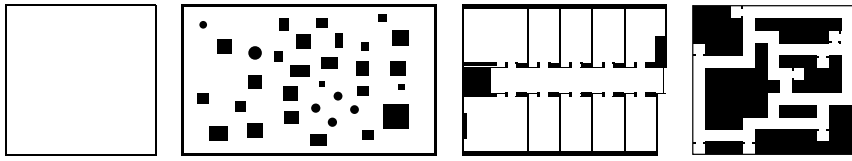


Figure 10. Maps used for the simulation experiments. From left to right: Empty environment, unstructured environment, office environment, and corridor environment

does not make a difference in situations like that depicted in a situation like that depicted in Figure 9. To minimize the completion time we therefore modify the evaluation function so that it considers squared distances to choose target locations $t_1, \ldots, t_n$:

$$\underset{(t_1,\ldots,t_n)}{\operatorname{argmax}} \sum_{i=1}^{n} \left[U(t_i \mid t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_n) - \beta \cdot (V_{t_i}^i)^2 \right]. \tag{5}$$

The resulting algorithm that determines in every round the optimal assignment of robots to target locations according to this evaluation function is given in Table 2. Compared to our greedy selection scheme, the major problem of this approach lies in the fact that in the worst case one has to figure out $\frac{m!}{(m-n)!}$ possible assignments where m is the number of possible target locations, n is the number of robots, and $m \leq n$. Whereas this number can be handled for small numbers of robots, it becomes intractable for larger numbers, because the number of possible assignments grows exponentially in the number of robots. In practice one therefore needs appropriate search techniques to find good assignments in a reasonable amount of time. In the experiments described here, we applied a randomized search technique with hill-climbing to search for optimal assignments.

To compare these three strategies we chose a set of different environments depicted in Figure 10. For each environment and each number of robots we per-

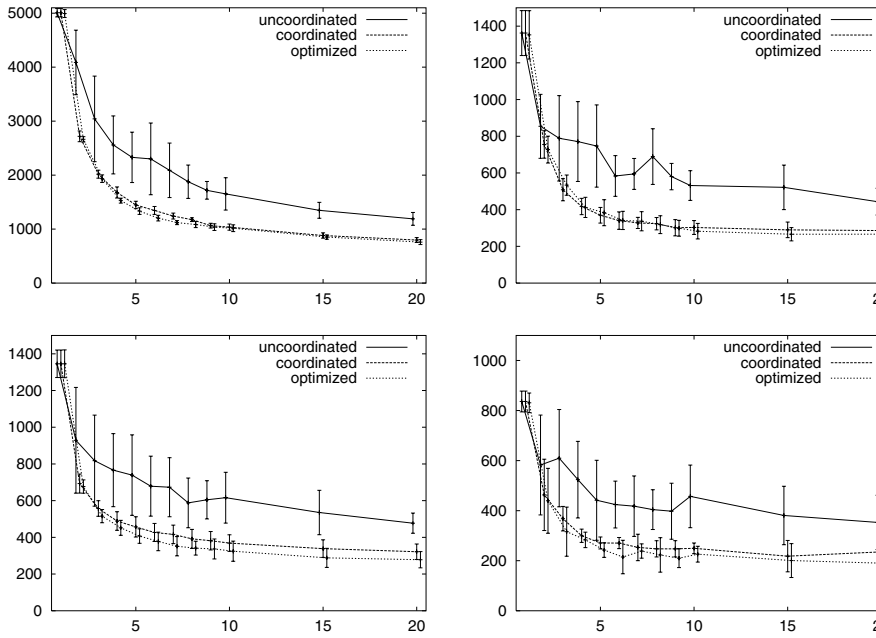


Figure 11. Performances of the different coordination strategies for the environments shown in Figure 10: Empty environment (top left), unstructured environment (top right), office environment (lower left), and corridor environment (lower right)

formed 8 different experiments. Thereby we varied over the points where the team was deployed at the beginning of each run. We then evaluated the average number of time steps the system needed to complete the job. The resulting plots are shown in Figure 11. The error bars indicate the 5% confidence level. As can be seen from the figure, the team using our algorithm significantly outperforms the uncoordinated system. It is worth noting that the optimization strategy on average is slightly better in the office environment and in the corridor environment, although the results are not significant.

These plots illustrate two advantages of our coordination approach. First, the robots are distributed over the environment so that they explore it much faster than uncoordinated robots. On the other hand, the robots are kept away from each other so that the number of interferences between them is minimized. For example, consider the results for 20 robots. In principle, coordination is less important the more robots are involved, since the probability that a certain area is explored quickly raises with the number of robots involved. However, if the robots are not distributed, the interferences between them result in longer execution times.

5 Related Work

The various aspects of the problem of exploring unknown environments with mobile robots have been studied intensively in the past. Different techniques for single robots have been presented in [32, 39, 51, 18, 23, 11, 16, 54, 50]. Whereas most of these approaches follow a greed strategy to acquire unknown terrain, they mainly differ in the way the environment is represented. Furthermore, there is a serious amount of theoretical work providing a mathematical analysis of the complexity of exploration strategies including comparisons for single robots [37, 31, 13, 14, 1, 2, 42]. Additionally [35] provides an experimental analysis of the performance of different exploration strategies for one mobile robot.

Also the problem of exploring terrains with teams of mobile robots has received considerable attention in the past. For example, Rekleitis et al. [43, 44, 45] focus on the problem of reducing the odometry error during exploration. They separate the environment into stripes that are explored successively by the robot team. Whenever one robot moves, the other robots are kept stationary and observe the moving robot, a strategy similar to [34]. Whereas this approach can significantly reduce the odometry error during the exploration process, it is not designed to distribute the robots over the environment. Rather, the robots are forced to stay close to each other in order to remain in the visibility range. Thus, using these strategies for multi-robot exploration one cannot expect that the exploration time is significantly reduced.

Cohen [12] considers the problem of collaborative mapping and navigation of teams of mobile robots. The team consists of a *navigator* that has to reach an initially unknown target location and a set of *cartographers* that randomly move through the environment to find the target location. When a robot discovers the goal point, the location is communicated among the cartographers to the navigation robot which then starts to move to the target location. In extensive experiments, the author analyzes the performance of this approach and compares it to the optimal solution for in different environments and different sizes of robot teams.

Koenig et al. [30] analyze different terrain coverage methods for ants which are simple robots with limited sensing and computational capabilities. They consider environments that are discretized into equally spaced cells. Instead of storing a map of the environment in their memory, the ants maintain markings in the cells they visit. The authors consider two different strategies for updating the markings. The first strategy is Learning Real-Time A* (LRTA*), which greedily and independently guides the robots to the closest unexplored areas and thus results in a similar behavior of the robots as in [53]. The second approach is Node Counting in which the ants simply count the number of times a cell was visited. The paper shows that Learning Real-Time A* (LRTA*) is guaranteed to be polynomial in the number of cells, whereas Node counting can be exponential.

Billard et al. [7] introduce a probabilistic model to simulate a team of mobile robots that explores and maps locations of objects in a circular environment. In several experiments they demonstrate the correspondence of their model with the behavior of a team of real robots.

In [4] Balch and Arkin analyze the effects of different kinds of communication on the performance of teams of mobile robots that perform tasks like searching for objects or covering a terrain. The "graze task" carried out by the team of robots corresponds to an exploration behavior. One of the results is that the communication of goal locations does not help if the robots can detect the "graze swathes" of other robots.

The technique presented in [33] is an off-line approach, which, given a map of the environment, computes a cooperative terrain sweeping technique for a team of mobile robots. In contrast to most other approaches this method is not designed to acquire a map. Rather the goal is to minimize the time required to cover a known environment which can lead to a more efficient behavior in the context of cleaning or mowing tasks.

Yamauchi et al. [53] present a technique to learn maps with a team of mobile robots. In this approach the robots exchange information about the map that is continuously updated whenever new sensor input arrives. They also use map-matching techniques [54] to improve the consistency of the resulting map. To acquire knowledge about the environment all robots follow a greedy strategy and move to the closest frontier cell. They are not applying any strategies to distribute the robots over the environment or to avoid that two or more robots explore the same areas.

One approach towards cooperation between robots has been presented by Singh and Fujimura [49]. This approach especially addresses the problem of heterogenous robot systems. During exploration each robots identifies "tunnels" to the so far unexplored area. If a robot is too big to pass through a tunnel it informs other robots about this tunnel. Whenever a robot receives such a message it either accepts this new task or further delegates it to smaller robots. In the case of homogeneous robots, the robots perform a greedy strategy similar to the system of Yamauchi et al. [53].

Furthermore, there has been several work focusing on the coordination of two robots. The work presented by Bender and Slonim [6] theoretically analyzes the complexity of exploring strongly-connected directed graphs with two robots. Roy and Dudek [46] focus on the problem of exploring unknown environments with two robots. Specifically, this paper presents an approach allowing the robots with a limited communication range to schedule rendezvous. The algorithms are analyzed analytically as well as empirically using real robots.

Finally, several researchers have focused on architectures for multi-robot cooperation. For example, Grabowski et al. [24] consider teams of miniature robots that overcome the limitations imposed by their small scale by exchanging mapping and sensor information. In this architecture, a team leader integrates the information gathered by the other robots. Furthermore, it directs the other robots to move around obstacles or to direct them to unknown areas. Jung and Zelinsky [29] present a distributed action selection scheme for behavior-based agents which has successfully been applied to a cleaning task. Matarić and Sukhatme [38] consider different strategies for task allocation in robot teams and analyze the performance of the team in extensive experiments.

In contrast to all approaches discussed above, the technique presented in this paper explicitely coordinates the actions of the robots so that they are distributed over the environment while they are exploring the environment. Accordingly the time needed to complete the exploration task is significantly reduced.

6 Summary and Conclusions

In this paper we presented a technique for coordinating a team of robots while they are exploring their environment. The key idea of this technique is to simultaneously take into account the cost of reaching a so far unexplored location and its utility. Thereby, the utility of a target location depends on the probability that this location is visible from target locations assigned to other robots. Our algorithm always assigns that target location to a robot which has the best trade-off between utility and costs. Our method differs from previous techniques in an explicit coordination mechanism that assigns different target locations to the robots. Some of the previous approaches to multi-robot exploration either forced the robots to stay close to each other or used a greedy strategy which assigns to each robot the target point with minimum cost. This, however, does not prevent different robots from selecting the same target location.

Our technique has been implemented and tested on real robots and in extensive simulation runs. Experiments presented in this paper demonstrate that our algorithm is able to effectively coordinate a team of robots during exploration. They further demonstrate that our coordination technique outperforms other methods developed so far.

Despite these encouraging results, there are several aspects which could be improved. One interesting research direction is to consider situations in which the robots do not know their relative positions. In this case the exploration problem becomes even harder, since the robots now have to solve two problems. On one hand they have to extend the map and on the other hand they need to find out where they are relative to each other. A further possible research direction is towards systems with a limited communication range. In this case the systems also have to plan rendezvous to exchange information and to avoid redundant operations.

Acknowledgments

This work is sponsored in part by the EC (contract number IST-2000-29456). The authors would furthermore like to thank Dieter Fox, Reid Simmons, and Sebastian Thrun for fruitful discussions.

References

[1] S. Albers and M.R. Henzinger. Exploring unknown environments. *SIAM Journal on Computing*, 29:1164–1188, 2000.

[2] S. Albers, K. Kursawe, and S. Schuierer. Exloring unknown environments with obstacles. In *Proc. of the 10th Symposium on Discrete Algorithms*, 1999.

[3] D. Apostolopoulos, L. Pedersen, B. Shamah, K. Shillcutt, M.D. Wagner, and W.R.L. Whittaker. Robotic antarctic meteorite search: Outcomes. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 4174–4179, 2001.

[4] T. Balch and R.C. Arkin. Communication in reactive multiagent robotic systems. *Journal of Autonomous Robots*, 1(1):27–52, 1994.

[5] R.E. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, 1957.

[6] M. Bender and D. Slonim. The power of team exploration: Two robots can learn unlabeled directed graphs. In *Proc. of the 35rd Annual Symposium on Foundations of Computer Science*, pages 75–85, 1994.

[7] A. Billard, A.J. Ijspeert, and A. Martinoli. A multi-robot system for adaptive exploration of a fast changing environment: Probabilistic modelling and experimental study. *Connection Science*, 11(3/4):357–377, 2000.

[8] W. Burgard, M. Moors, D. Fox, R. Simmons, and S. Thrun. Collaborative multi-robot exploration. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2000.

[9] Y.U. Cao, A.S. Fukunaga, and A.B. Khang. Cooperative mobile robotics: Antecedents and directions. *Autonomous Robots*, 4, 1997.

[10] J.A. Castellanos, J.M.M. Montiel, J. Neira, and J.D. Tardós. The SPmap: A probabilistic framework for simultaneous localization and map building. *IEEE Transactions on Robotics and Automation*, 15(5):948–953, 1999.

[11] H. Choset. Topological simultaneous localization and mapping (slam): Toward exact localization without explicit localization. *IEEE Transactions on Robotics and Automation*, 17(2), April 2001.

[12] W. Cohen. Adaptive mapping and navigation by teams of simple robots. *Journal of Robotics and Autonomous Systems*, 18:411–434, 1996.

[13] X. Deng, T. Kameda, and C. Papadimitriou. How to learn in an unknown environment. In *Proc. of the 32nd Symposium on the Foundations of Comp. Sci.*, pages 298–303. IEEE Computer Society Press, Los Alamitos, CA, 1991.

[14] X. Deng and C. Papadimitriou. How to learn in an unknown environment: The rectilinear case. *Journal of the ACM*, 45(2):215–245, 1998.

[15] G. Dissanayake, H. Durrant-Whyte, and T. Bailey. A computationally efficient solution to the simultaneous localisation and map building (SLAM) problem. Working notes of ICRA 2000 Workshop W4: Mobile Robot Navigation and Mapping, April 2000.

[16] G. Dudek, M. Jenkin, E. Milios, and D. Wilkes. Robotic exploration as graph construction. *IEEE Transactions on Robotics and Automation*, 7(6):859–865, 1991.

[17] G. Dudek, M.. Jenkin, E. Milios, and D. Wilkes. A taxonomy for multi-agent robotics. *Autonomous Robots*, 3(4), 1996.

[18] T. Edlinger and E. von Puttkamer. Exploration of an indoor-environment by an autonomous mobile robot. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 1994.

[19] H. Endres, W. Feiten, and G. Lawitzky. Field test of a navigation system: Autonomous cleaning in supermarkets. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 1998.

[20] M. Fontan and M. Mataric. Territorial multi-robot task division. *IEEE Transactions on Robotics and Automation*, 14(5), 1998.

[21] D. Fox, W. Burgard, H. Kruppa, and S. Thrun. Collaborative multi-robot localization. In *Proc. of the 23rd German Conference on Artificial Intelligence*. Springer Verlag, 1999.

[22] D. Goldberg and M. Mataric. Interference as a tool for designing and evaluating multi-robot controllers. *Journal of Robotics and Autonomous Systems*, 8:637–642, 1997.

[23] H.H. González-Baños, E. Mao, J.C. Latombe, T.M. Murali, and A. Efrat. Planning robot motion strategies for efficient model construction. In *Proc. Intl. Symp. on Robotics Research (ISRR)*, 1999.

[24] R. Grabowski, L.E. Navarro-Serment, C.J.J. Paredis, and P.K. Khosla. Heterogeneous teams of modular robots for mapping and exploration. *Journal of Autonomous Robots*, 8(3):293–308, 2000.

[25] D. Guzzoni, A. Cheyer, L. Julia, and K. Konolige. Many robots make short work. *AI Magazine*, 18(1):55–64, 1997.

[26] D.F. Hougen, S. Benjaafar, J.C. Bonney, J.R. Budenske, M. Dvorak, M. Gini, H. French, D.G. Krantz, P.Y. Li, F. Malver, B. Nelson, N. Papanikolopoulos, P.E. Rybski, S.A. Stoeter, R. Voyles, and K.B. Yesin. A miniature robotic system for reconnaissance and surveillance. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2000.

[27] R.A. Howard. *Dynamic Programming and Markov Processes*. MIT Press and Wiley, 1960.

[28] Y. Huang, Z.Cao, S. Oh, E. Kattan, and E. Hall. Automatic operation for a robot lawn mower. In *SPIE Conference on Mobile Robots*, volume 727, pages 344–354, 1986.

[29] D. Jung and A. Zelinksy. An architecture for distributed cooperative planning in a behaviour-based multi-robot system. *Journal of Robotics and Autonomous Systems*, 26(2-3):149–174, 1999.

[30] S. Koenig, B. Szymanski, and Y. Liu. Efficient and inefficient ant coverage methods. *Annals of Mathematics and Artificial Intelligence*, In Print.

[31] S. Koenig, C. Tovey, and W. Halliburton. Greedy mapping of terrain. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2001.

[32] B. Kuipers and Y.-T. Byun. A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations. *Journal of Robotics and Autonomous Systems*, 8:47–63, 1991.

[33] D. Kurabayashi, J. Ota, T. Arai, and E. Yoshida. Cooperative sweeping by multiple mobile robots. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 1744–1749, 1996.

[34] R. Kurazume and N. Shigemi. Cooperative positioning with multiple robots. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 1994.

[35] D. Lee and M. Recce. Quantitative evaluation of the exploration strategies of a mobile robot. *International Journal of Robotics Research*, 16(4):413–447, 1997.

[36] J.J. Leonard and H.J.S. Feder. A computationally efficient method for large-scale concurrent mapping and localization. In J. Hollerbach and D. Koditschek, editors, *Proceedings of the Ninth International Symposium on Robotics Research*, 1999.

[37] S. Lumelsky, S. Mukhopadhyay, and K. Sun. Dynamic path planning in sensor-based terrain acquisition. *IEEE Transactions on Robotics and Automation*, 6(4):462–472, 1990.

[38] M.J. Matarić and G. Sukhatme. Task-allocation and coordination of multiple robots for planetary exploration. In *Proc. of the International Conference on Advanced Robotics*, 2001.

[39] S.J. Moorehead, R. Simmons, and W.L. Whittaker. Autonomous exploration using multiple sources of information. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2001.

[40] M. Moors. Koordinierte Multi-Robot Exploration. Master's thesis, Department of Computer Science, University of Bonn, 2000. In German.

[41] H.P. Moravec. Sensor fusion in certainty grids for mobile robots. *AI Magazine*, pages 61–74, Summer 1988.

[42] N. Rao, S. Hareti, W. Shi, and S. Iyengar. Robot navigation in unknown terrains: Introductory survey of non-heuristic algorithms. Technical Report ORNL/TM-12410, Oak Ridge National Laboratory, 1993.

[43] I. Rekleitis, G. Dudek, and E. Milios. Multi-robot exploration of an unknown environment, efficiently reducing the odometry error. In *Proc. of International Joint Conference in Artificial Intelligence (IJCAI)*, 1997.

[44] I. Rekleitis, G. Dudek, and E. Milios. Accurate mapping of an unknown world and online landmark positioning. In *Proc. of Vision Interface (VI)*, 1998.

[45] I. Rekleitis, R. Sim, G. Dudek, and E. Milios. Collaborative exploration for the construction of visual maps. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2001.

[46] N. Roy and G. Dudek. Collaborative robot exploration and rendezvous: Algorithms, performance bounds and observations. *Journal of Autonomous Robots*, 11(2):117–136, 2001.

[47] R. Simmons, D. Apfelbaum, W. Burgard, D. Fox, M. Moors, S. Thrun, and H. Younes. Coordination for multi-robot exploration and mapping. In *Proc. of the National Conference on Artificial Intelligence (AAAI)*, 2000.

[48] M. Simoncelli, G. Zunino, H.I. Christensen, and K. Lange. Autonomous pool cleaning: Self localization and autonomous navigation for cleaning. *Journal of Autonomous Robots*, 9(3):261–270, 2000.

[49] K. Singh and K. Fujimura. Map making by cooperating mobile robots. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 254–259, 1993.

[50] A. Stentz. Optimal and efficient path planning for partially-known environments. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3310–3317, 1994.

[51] C.J. Taylor and D.J. Kriegman. Exloration strategies for mobile robots. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 248–253, 1993.

[52] S. Thrun. A probabilistic online mapping algorithm for teams of mobile robots. *International Journal of Robotics Research*, 20(5):335–363, 2001.

[53] B. Yamauchi. Frontier-based exploration using multiple robots. In *Proceedings of the Second International Conference on Autonomous Agents*, pages 47–53, 1998.

[54] B. Yamauchi, A. Schultz, and W. Adams. Integrating exploration and localization for mobile robots. *Adaptive Systems*, 7(2), 1999.

Mental Models for Robot Control

Hans-Dieter Burkhard, Joscha Bach, Ralf Berger,
Birger Brunswieck, and Michael Gollin

Humboldt University, Unter den Linden 6, 10099 Berlin, Germany
{hdb,bach,berger,brunswie,gollin}@informatik.hu-berlin.de

Abstract. The paper investigates problems of real time control for complex long term behavior in dynamically changing environments, especially with regard to requirements arising in actual applications. The scenario of robotic soccer (RoboCup) is used for illustration. The proposed Double Pass Architecture avoids some difficulties of layered hybrid architectures. It allows for real time adaptations to new situations even on the higher levels. It implements concepts of bounded rationality, and supports Case Based Reasoning methods.

1 Introduction

Autonomous robots that act on dynamic environments need capabilities for deliberation and reactive behavior. Hybrid architectures have been developed for indoor service robots and for unmanned ground vehicles (UGV). The underlying assumptions concern stable long term goals and fast reactions for intermediate exceptions like obstacle avoidance. For this reason, real time requirements apply only to the basic behaviors.

In this paper, we investigate the problems of real time requirements for complex long term behavior, for example on the coordination level in layered architectures. Fast changes of coordinated behaviors are needed for teams of rescue robots, for example. We will use the scenario of robotic soccer [Kitano-et-al-97], [RoboCup] for illustration.

The domain of robotic soccer is an interesting application for robotic control architectures. Because of some of its properties, like incomplete, sometimes incoherent information, restricted computational resources, the need to plan and act in real time, it requires relatively specialized agent frameworks that are both flexible and efficient.

Our group is participating in two leagues of RoboCup: the Sony Four Legged Robotic League, and the Simulation League. The Sony robot competitions are characterized by the fact that the hardware is fixed (i.e. can not be extended with additional processing power, memory or sensors). These constraints apply equally to all teams and provide a need to concentrate on software solutions that can exploit the limits of the platform, rather than adding more sophisticated hardware to overcome problems with sensing, processing and actorics. The Simulation League is far less concerned with the specifics of sensorics and actorics. Hence the challenges are cooperative action, strategic behavior, opponent modeling, learning.

M. Beetz et al. (Eds.): Plan-Based Control of Robotic Agents, LNAI 2466, pp. 71–88, 2002.

We find that the different demands of these leagues lead to a very different focus during development. However, both systems have hierarchical architectures with similar high level control demands. We have found that planning should be performed in a more open style, and we propose to use Case Based Reasoning (CBR) for mental models.

Experiences of all teams in RoboCup point to problems with scalability concerning time horizon and variety of behavior. We argue that these problems follow from certain restrictions of classical hybrid architectures. We have devised a control architecture which is able to exploit CBR methods and fulfills the demands of long-term planning and real time re-planning of high level goals. It uses ideas of bounded rationality [Bratman87] for restricting the scope of decisions. Since the control is splitted into two separated top-down passes, we call it a "Double Pass Architecture". Programs are based on roles and can be specified in an XML based language. The architecture has been implemented on both platforms.

The paper is organized as follows: We start with a discussion of requirements for robot control and planning in dynamically changing environments. Then we examine the question how planning in RoboCup could look like. We investigate, how different alternatives of action increase the demands to the architecture dramatically, if the time horizon is enlarged. A short summary concerning the dimensions of control architectures follows, providing the basic requirements of architectures for RoboCup robots. The Double Pass Architecture is described in the last section before the conclusions.

The authors like to thank the previous and recent members of the teams "AT Humboldt" (Simulation League) and "German Team" (Sony Four Legged Robotic League) for a lot of fruitful discussions. The paper could not have been written without their theoretical and practical work. The work is granted by the German Research Association (DFG) in the main research program 1125 "Cooperating teams of mobile robots in dynamic and competitive environments".

2 Robot Control and Planning in Dynamically Changing Environments

Control of autonomous robots in dynamic environments is interesting from a cognitive point of view as well as under application view points. Some of the different approaches are influenced by examples from nature, including

1. Simple, but well tuned reactions to inputs from the real world. Complex behavior can emerge by only immediate reactions to the environment without any internal (symbolic) modeling or planning. The complexity of the environment is exploited for control: "The best model of the world is the world itself". The classical example is obstacle avoidance without complex models and plans.
2. Actions following long term plans using complex internal models, explicit goals and plans. There are different approaches ranging from state space

search to logics and different planning paradigms. Such approaches are necessary for more complex behavior exploiting symbolic representations like path planning using a map.
3. Swarm intelligence where complex behavior emerges from the interaction of large groups of simple agents. This approach can be seen as an extension of the first one, where individual agents monitor each other and adjust their behavior accordingly.

We will consider the first two approaches in the paper. Detailed discussions can be found in textbooks like [Arkin1998,Dudek00,Murphy00], for the third approach we refer to [Parunak97].

Hybrid architectures combine the first two approaches. They are organized as layered architectures with low level reflex behaviors and high level planning capabilities. They have been developed e.g. for indoor service robots and for unmanned ground vehicles. Low level behaviors deal with real time requirements like motion commands. They provide for safety in dynamically changing environments. Preconditions and effects of low level behaviors are described in a symbolic fashion and may be subject to combination, arbitration, state space search, and planning. Special techniques like potential fields are used for guidance of low level behavior. Recent robots in RoboCup use related control structures, but a closer look shows that the classical architectures have certain limitations. Complex coordinated behavior like wing changes or double passes do emerge sometimes by chance, but they do not occur intentionally. Interestingly, there is a revival of the old debate if emergent behavior is sufficient or if deliberation is really needed. We argue that deliberation is necessary (and possible).

As an example, let us consider the following situation from RoboCup: The offensive team wants to change wings over several stations. The players involved in that maneuver try to find free positions (e.g. with distances to opponents) for the successive passes. If one of these passes is catched by opponents, then the team has to switch to defensive play as fast as possible. The players do not have to maintain distance to opponents anymore, now they have to get close to them for covering. Useful time is lost if the switch to new behaviors is delayed. The switch concerns the higher control level (from offensive to defensive). Here we find a problem of hybrid architectures, since the higher levels are usually called with lower frequencies. This is useful for UGVs since deliberation needs more time. It is sufficient since the results of deliberations are more stable under dynamics (cf. "local dynamics" below).

Here are some of the differences between RoboCup robots and simple indoor robots or 'classical' UGVs:

– Many UGVs can control their behavior entirely without examining the expected consequences of their actions. This corresponds to simple movements of the players in RoboCup. Actually, the players use similar strategies of obstacle avoidance during positioning.
  The intrinsic subject of control is the ball, but it can be controlled only from time to time. Actions of players are concerned with reaching/defending the control over the ball by the team (not only individually).

- Often, an UGV has to reach certain fixed locations. The locations are determined by planning as intermediate goals in order to reach final goals. Classical planning is applicable to determine an appropriate route.
  One may theoretically come up with a plan to play the ball via several players from the goal-kick to the opponents goal, but it is hard to imagine such a plan to work in reality. This is a great difference to a chess program: Here, it is relatively easy to write a program for finding the ultimate best moves, it is "only" a question of complexity to run this program.
- The number of behaviors (e.g. different methods for kicking and dribbling of a legged robot) is much larger than usually considered for UGVs. It is not clear, how far scaling of known methods is possible when there are hundreds of different behaviors.
- Dynamics of the UGV environment are typically of a local kind. They concern occasional changes of the paths, e.g. for obstacle avoidance. Variations are temporary and intermediate, while the final goals remain unchanged.
  We call this *local dynamics*, they can be handled on the lower level of a hybrid architecture.
  The situation is quite different for RoboCup: The loss of control of the ball forces a change on a higher level from offensive to defensive play (with related consequences for the lower levels).
  We call this *total dynamics*; they may be caused by intervention of the referees, by loss of ball control and by unexpected behavior of other players. As an example we have mentioned the wing change maneuver above. Total dynamics cannot be handled by the lower levels of a hybrid architecture alone.
- Sometimes, UGVs are controlled by potential fields (*'force fields'*), which define preferred directions of the vehicle according to a precomputed plan. Several teams in RoboCup have experimented with potential fields for possible ball movements: Team mate positions are attractors for the ball, opponents are considered as repulsors.
  While potential fields for UGVs are mostly static, the ball related potential fields in RoboCup change rapidly with the movements of players. Hence potential fields are not as useful for path planning for the ball. Potential fields can be used in another way: Profitable changes of potential fields provide useful guidance for positioning of players.

3 Planning for RoboCup Robots?

The question arises if there is a possibility for planning in RoboCup at all. Of course, STRIPS like planners are not useful because of total dynamics and adversaries. The recent experiences in RoboCup show that teams try to improve their short term behaviors (skills), but make only small efforts for long term deliberation. Coordination is maintained using small subgroups (formations) which mainly maintain collocation.

The control of soccer robots uses complex world models where the sensor information of different players are merged and where simulation (for example

of ball movements) can be performed. They players analyze situations using e.g. opponent models, simulation, utilities (cf. Section 4). Appropriate behaviors (action sequences) on the level of kicking/passing, dribbling, intercepting, tackling, positioning are selected and performed. Complex maneuvers are not planned and then followed through but rather emerge by chance.

These architectures do not really fit into the classical scheme of reactive vs. deliberative control as described in [Murphy00,Arkin1998,Dudek00]: It is not really deliberative (there is no long term planning), and it is not really reactive (there is a complex world model and there are complex symbolic computations). We will come back to this point in Section 4.

The problems concerning emergent complex behaviors from simple ones is mostly discussed as it has been ten years ago: Do we need deliberation, is planning useful for environments with such dynamics etc.? Looking for humans we see that there is in fact proactivity concerning long term behavior. Human players use intentionally so-called standard situations. They concern coordinated play for standard situations like corner-kick, throw-in, wing change, double pass, offside trap etc. The behavior in these situations is describable on a symbolic level. Additionally, experience in common training and common matches forms a team: team mates know about each other. Players learn anticipation and coordination of complex behavior of team mates and act accordingly.

What are these behaviors like? They are not low level, and are not realized using sensor-actor-coupling. They are on a symbolic level, and can be composed from other (sub-)behaviors, i.e. they can be described by hierarchical structures. They are not classical plans, but can be understood as conditional partial plans in the sense of least commitment strategies. For example, a wing change consists of several changes of positions and successive passes between players. The specification of a pass with concrete parameters (direction, power) depends on the appropriate position of a team mate. Least commitment means that only the involved players and their roles are defined in the beginning of the change wings behavior. The specification of a pass is postponed until the appropriate situation is reached.

The analysis and identification of such appropriate situations is another crucial point. The wing change behavior sets a context for that analysis which restricts on-line decisions to a limited scope. The notion "appropriate" situation refers to some kind of precondition for the pass.

Preconditions are requisites for the combination of (sub-)behaviors ("operators"). The difference to classical planning is their indetermination and vagueness. The sequence of (sub-)behaviors is in principle predefined by the behavior 'wing change'. The players are forced to create appropriate preconditions for the intended (sub-)behavior step by step (e.g. by permanent efforts for good positioning). These preconditions can be fulfilled in different ways depending on the concrete situation, preferences of players and coordination "patterns".

An attempt to model such kind of deliberation has to model related behaviors and decision strategies. We are working on a project to use case based reasoning ([Kolodner,Lenz-et-al98]): Complex behaviors, standard situations, player pref-

erences etc. can be considered as cases and they can be stored in a case base. Recent situations are analyzed and checked by the players for their similarity to cases from the case base. The behavior of a similar case is then adapted.

Good play depends on opponent modeling. We have experimented with simple models [WendlerLenz98] using CBR. The problem with models is again the dynamic behavior: When our players adapted to the recognized behavior, then the opponents reacted immediately with another behavior. We had analyzed "preferred" positions of free opponents and tried to keep closer to these places. But since opponents tried to keep distance, they moved to other places. Actually, our simple assumption of (fixed) "preferred" positions was wrong. Successful models of opponents should assume a more complex control structure.

For the realization of CBR approaches we need a case format which can describe hierarchically structured complex behavior. We need an architecture to handle cases, similarity and case based control. The architecture should satisfy the needs of RoboCup robots.

4 Alternatives for Control in RoboCup

The football/soccer scenario provides a lot of situations to illustrate the requirements and alternative solutions concerning control architectures for dynamic environments. We start with some basic behaviors.

4.1 Simple Behavior: Ball Interception

- The information provided by sensors is incomplete (the ball may be covered by other players) and imprecise (due to noisy data). The robot may hence use a model of the environment ("world model") to store information received in the past. The world model is updated according to new information. Such a world model is a *persistent state*. Persistency means that information is kept and used later for building a new world model with newer sensory data.
- Interception of a moving ball illustrates simple problems of the dynamic environment: A very simple "stimulus-response player" would run straight to the place where he sees the ball. As the ball is moving he has to adjust his direction every time he looks for the ball, and he will perform a curved path as the result. A more skillful player could anticipate the optimal point for interception and run directly to this point.
- Using the same example of the moving ball, we may think about the procedure for anticipation. The robot calculates the speed vector v for the optimal run to the ball depending on the recent position p and the speed u of the ball (relative to the player). It may use additional parameters according to opponents, weather conditions, noise etc.
  The calculation may explicitly exploit physical laws (including e.g. the expected delay of the ball). It may use simulation (forward model) for possible speed vectors v of the player. If an inverse model is available, the optimal speed vector v may be calculated directly. Determination of v may use a

look-up table with precomputed values or a neural network which has been trained by real or simulated data.
- We still consider the optimal interception of a moving ball using calculations of the speed vector v. The calculation can be repeated whenever new sensor information is available. Thus, it always can regard newest information and hopefully obtain the best speed vector v. Alternatively, the player may keep moving according to v for a longer time. Therefore he needs another kind of *persistent state* to memorize this goal.
  If the ball is not observable for some time (e.g., if it is covered by another player), then the persistent goal is used as the trigger to keep running. Alternatively, simulating the ball in the world model can also be a trigger to continue the interception process.
- Problems with the reliability of the computed speed vector v arise due to noise in the sensory data (and may be due to imprecise calculations themselves). Repeated calculations may hence result in oscillations and suboptimal behavior (as reported in [Müller-Gugenberger and Wendler, 1998]). It may be better to follow the old speed v_t as long as the difference to the new speed v_{t+1} is not too large (hysteresis control). Exploiting the inertia of the robot provides another way using the physical world directly.

The discussion shows a lot of different approaches and implementations for the simple behavior "follow a moving object". In most cases there is a lot of redundancies which can be exploited for efficient and more reliable controls in different ways yielding different trade offs. Since single methods are often of restricted reliability, the appropriate combination (regarding the overall system) is a challenging design problem.

4.2 Coordination

More complex problems of dynamic environments are illustrated by coordination. Control becomes more and more complex as the time horizon is enlarged. An increasing diversity of approaches is inherited from the underlying methods and additionally caused by different planning methods. Here are some examples of control on an intermediate level of complexity:

- A player decides if he can intercept the ball, i.e. if the ball is reachable. The decision process can use the procedures for computing v from above to calculate the interception point and time.
- A player decides if he can intercept the ball before any other player. Therefore he has to compare with the interception times of other players (e.g. using the methods to calculate v from the view point of other players).
- A player decides not to intercept the ball even if he is the first to reach the ball. He may leave the ball for a team mate in a better position for continuation.

A player controlling the ball has different alternatives. He can try to score, he can pass to a team mate, he can dribble or kick simply far away for certain

reasons. The decision can be based on calculations with different complexity. We consider the analysis for a pass:

- The player could calculate which of his team mates can reach an appropriate pass. This can be computed by simulations using the interception calculations from above: Given a hypothetical kick (ball speed v), which players would reach the ball first? Testing different speed vectors v leads to preferences for different passes. The best scoring pass is performed.
- The above procedure prefers a pass to a player which can best reach the ball. Then the question arises if this player has good chances for continuation. Complete simulations including larger time horizons are impossible because of the dynamics. Instead, the scoring by the above procedure can be biased according to further criteria (forward direction, chances to score, players in the neighborhood etc.).
- If the pass is embedded in a larger behavior, then the situation is quite different. We consider again the wing change example from above. Now the pass is bound to the overall behavior, it should reach a team mate engaged in that maneuver. Moreover, the player can trust in cooperative behavior (according to roles), and he can wait until the team mate is prepared (has established preconditions, i.e. reached a free position).

Optimal behavior of the players not controlling the ball is an important factor of successful play. It has to establish alternatives for future play while restricting the chances of opponents. Especially this behavior is hard to guide by simple strategies only. Long term strategic considerations are important. Human players use predefined behavior patterns for coordination as discussed above. State of the art in RoboCup are strategies for positioning according to the most recent situation, especially following the movements of the ball. Such strategies include

- Positioning for standard situations like goal kick.
- Positioning regarding off-side lines.
- Positioning using potential fields.
- Positioning using fixed patterns.
- Positioning using subteams.

Some teams in Simulation League use a coach for the analysis of position play (trying to discover the strategy of the opponents). The coach is allowed to give advice during breaks. No attempts have been seen (as far as we know) to implement long term coordinated positioning.

5 Dimensions of Control Architectures

The classical distinction between control architectures concerns two extremes (cf. e.g. [Brooks91,Maes90,Georgeff-Lansky87,JPMüller96,Arkin1998,Murphy00], [Dudek00]).

- Reactive architectures without any persistent state (no world model, no commitment), realizing direct coupling between sensors and actors. There is no symbolic reasoning. This is understood as *simple behavior*.
- Deliberative architectures are identified with complex reasoning capabilities using persistent states (world model, commitment), symbolic reasoning and (classical) planning methods.

Planning was originally understood as state space search, which was appropriate for micro worlds (like blocks world scenarios). Other techniques have been introduced later for more complex and dynamic environments, standard examples are indoor service robots and UGVs. Thereby, deliberative control is still thought to be not time critical. If real-time constraints have to be considered, reactive approaches are used. Hybrid architectures are used to combine deliberative planning needs with reactive behaviors. Layered architectures provide a structure where real-time constraints are maintained by the lower (reactive) layers, and the deliberative layers are invoked with a lower frequency.

	world model (state)	commitment (state)	deliberation	symbolic	sensor-actor coupling
reactive	no	no	no	no	yes
deliberative	persistent	persistent	yes	yes	no
hybrid	persistent	persistent	yes	yes	yes
SRwWM	persistent	no	no	yes/no	no
chess program	(persistent)	no	yes	yes	no

Fig. 1. Some Architectures (SRwWM = Stimulus Response with World model as in [RussellNorvig95])

Actually, this classification is very rough. Figure 1 shows some aspects combined in the notions of reactive and deliberative robots. An important point of classification concerns the existence of persistent states for modeling the outside world (keeping "information from the past") and/or persistent states for goals and plans (keeping "information concerning commitments for the future") as discussed in [Burkhard2001]. Important dimensions of control include:

- Direct coupling of sensors and actors (e.g Braitenberg vehicles).
- Perception: Symbolic representation of the outside world using sensor fusion, background knowledge, merging with data from other robots.
- Existence of a persistent world model storing the results of perception (persistent states concerning the past).
- Deliberation: Decision procedures using e.g. anticipation (simulation) of possible future developments, goal orientation, planning, utilities etc.
- Existence of a persistent commitment storing the results of deliberation (persistent states concerning the futures).
- Complexity of computations.
- Centralized vs. distributed states and computations.

As an example we may consider a chess program: It can anticipate future situations considering the possible moves of both players starting with the recent situation. It can evaluate reachable future "goals" using complicated evaluation procedures. Finally it comes up with simply the next move, and all intermediate results are forgotten. After the opponent's move, the same process starts again for the new situation without any reference to the previous computations. It maintains complex decision procedures but it does not have any persistent states for commitments. It may have a world model which is updated by moves. But it would make no essential difference to input (to "sense") the whole board at each cycle. Then it would fulfill most definitions of reactive controls except for the complex symbolic deliberation.

By persistency we mean keeping information for later computations. Therewith, the notion depends on the meaning of "later". A discrete control cycle is considered for sake of simplicity. Since the actions of a robot depend on previous sensor information, the so-called "sense-think-act"-cycle is a reasonable model of control flow. There is some freedom for choosing the time steps $t_1, t_2, t_3, \ldots$, and we identify the time steps with the arrival of sensory data ("input") at the control unit. Now the notion of "persistency" depends on this definition: We call a state a persistent state if it keeps information from one step to the next. The chess program considered above does not have persistent states since it needs no persistent world model, and computed goals are forgotten after a move is performed.

World models, goals and plans of a RoboCup robot are not persistent as long as they cannot be used by the decision process in the next time step. Our implementation of the control architecture in [Burkhard *et al.*, 1998] was based on the notions of belief, desire and intentions (BDI, cf. [Bratman87,RaoGeorgeff91]) to describe intermediate results. But the concepts did not stand for persistent states since the decision process was started from the beginning in each time step. Since some stability problems occurred, some reference to the old intention was introduced to maintain stability later.

The cycle of a deliberative robot can be described by Figure 2. Simpler structures are obtained if we omit persistency. For example, without a persistent commitment state, i.e. for `commitment`$_t$ `:= deliberate(worldmodel`$_t$`);` we get the "stimulus-response-architecture with worldmodel" as described in [RussellNorvig95].

```
for t = 0,1,2,...  do
       worldmodel_t := perceive(worldmodel_{t-1}, input_t);
       commitment_t := deliberate(commitment_{t-1} , worldmodel_t);
       output_t := execute(commitment_t);
```

Fig. 2. Architecture with Persistent States for Worldmodel and Commitment

Hybrid architectures are covered by this scheme, at least in principle. They have different time scales since the deliberation layer works with lower frequency. This means that the synchronization between `deliberate` and `execute` is somewhat different to the description by Figure 2. The commitment can be understood as a (may be conditional) plan or script computed on "higher levels" (by `deliberate`) at time t such that

$\texttt{commitment}_t$ = $\texttt{step}_t$, $\texttt{step}_{t+1}$,..., $\texttt{step}_{t+k}$.

The "low level" `execute`-function computes the outputs for $i = t, \ldots, t + k$ according to that script:

$\texttt{output}_i$:= execute($\texttt{step}_i$, $\texttt{input}_i$) .

The most recent input $\texttt{input}_i$ (or the world model if available in time) is used for adaptation. A temporary "reactive behavior" is realized if identical steps $\texttt{step}_i$ are used. As an example we may think of the commitment to run to a certain position, where the `execute`-function has to realize movement steps over a longer time.

While `execute` is active at each time step t, the $\texttt{commitment}_t$ remains unchanged over longer time intervals in the layered hybrid architecture. During this time, the next commitment $\texttt{commitment}_{t+k+1}$ is computed.

6 Requirements for Architectures of RoboCup Robots

We have described several differences between mobile robots like UGVs and RoboCup robots in section 2. A very important one concerns the different time constraints (RoboCup robots must be able to revise their goals up to the highest control level in real-time because of total dynamics). Further important differences concern the character of high level behaviors and planning paradigms.

In addition to low level behaviors, we want to have complex high level behaviors which are subject to symbolic descriptions and which are performed using symbolic computations. High level behaviors should be composed from lower ones. The approach should be scalable with respect to large numbers of behaviors and with respect to additional levels in the hierarchy (more complex plans).

Classical plans as fixed operator sequences with fixed static preconditions and postconditions are not useful under the dynamics of RoboCup. There are compositions of (sub-)behaviors, but the transition from one to the next depends on dynamical conditions as discussed in Section 3. Some of these limitations can be overcome by using methods from CBR, which must include least commitment aspects as discussed in Section 3.

Scalability is an important requirement. It concerns e.g. larger time horizon for deliberation, larger numbers of behaviors, more parameters of behaviors, coordinated behavior.

The following points are relevant for our attempts for a control architecture of RoboCup robots:

- Total dynamics: real-time conditions apply on all levels.
- Hierarchically structured behaviors. Behaviors can be composed to more complex ones including coordination between players.
- Symbolic description of (high level) behaviors. Goal-directed approaches like the BDI-approach ([RaoGeorgeff91,Bratman87]) are useful.
- Large number of different behaviors (e.g. different kinds of dribbling, kick, intercepts as provided by legged robots). Contexts set by higher levels are useful.
- Least commitment.
- Handling of multiple goals.
- Maintenance of stability for coordination. (For problems behind stability and flexibility cf. e.g. [Bratman87,Burkhard00]).
- Description of standard situations by cases.
- Analysis of situations and decision making using CBR techniques.
- Adaptation via off-line and on-line learning, including modeling of opponents.

We have implemented an architecture which hopefully deals with these problems in a reasonable way. Programs are specified in an XML based language. To deal with total dynamics, we permit goal changes on the highest levels in real-time, i.e. using the time scale of the execution process. We have implemented a so-called *executor* which passes in real-time over all levels. Since changes on higher levels would overwrite changes on lower levels, the control flow is performed top down.

The real-time requirements cannot be realized in procedures for longer, time consuming deliberations. Hence we have implemented a so-called *deliberator* for complex analysis of situations and preparing high level behavior for execution. Again, the deliberation on high level determines the deliberation on the lower level, and the deliberator passes top down over all levels.

Both passes are independent from each other, hence the architecture is called Double Pass Architecture.

7 The Double Pass Architecture

The architecture uses persistent states for the past (world model) and for the future (desires, intentions). It implements goal-directed behavior inspired by the BDI-approach.

It uses a hierarchical structure and a least commitment strategy. The main idea is to distinguish between two *passes* of operation:

The Deliberator performs the choice of intentions and the longterm planning. It sets up a partial hierarchical plan. Following the least commitment idea, the plan is refined as time goes on. The deliberator does not have time problems since he works with sufficient forerun. Time critical decisions are left to the executor.

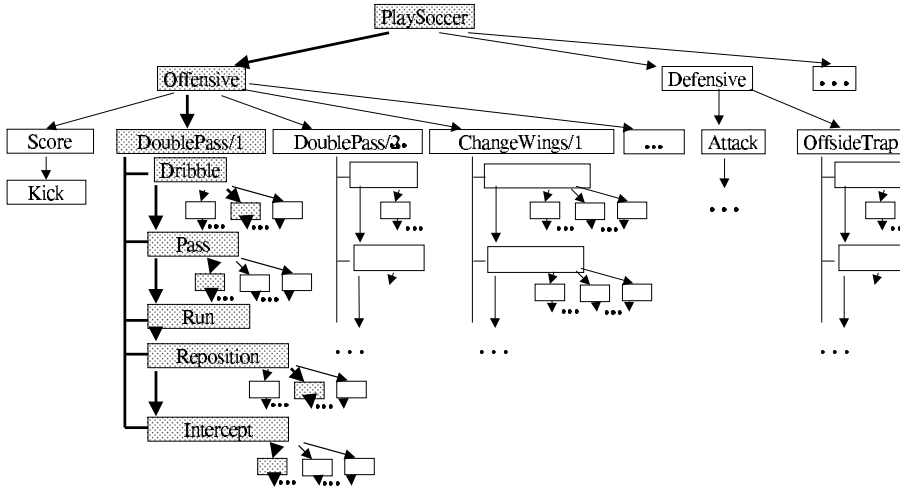


Fig. 3. Option Tree with Intention Subtree

The Executor performs just-in-time decisions. Based on the preparatory work of the deliberator, its choices are restricted to a minimum of decisions which need the most recent sensory information.

Both passes are independently running through all(!) levels of the hierarchy: Thus we have a *"double pass"* runtime structure. This is in contrast to runtime organizations in classical layered architectures (where short term decisions only affect the lowest level) and in programming languages (where only the procedure on the top of the stack is active). The distribution of utility/feasibility evaluations and condition checks over the different layers makes it possible to associate the individual levels of planning and action decisions with the respective architectural layers.

7.1 Options

The data structure where desires and intentions are chosen from are the *options*. We do not call them behaviors in order to emphasize their symbolic character and to relate it to desires and intentions. The set of options can be considered as a (virtual) tree structure with long term options near the root and specific short term actions near the leafs. An example from the soccer domain is given in Figure 3.

An option is processed by performing appropriate subintentions as defined by the tree. There are two kinds of connections between options and subintentions:

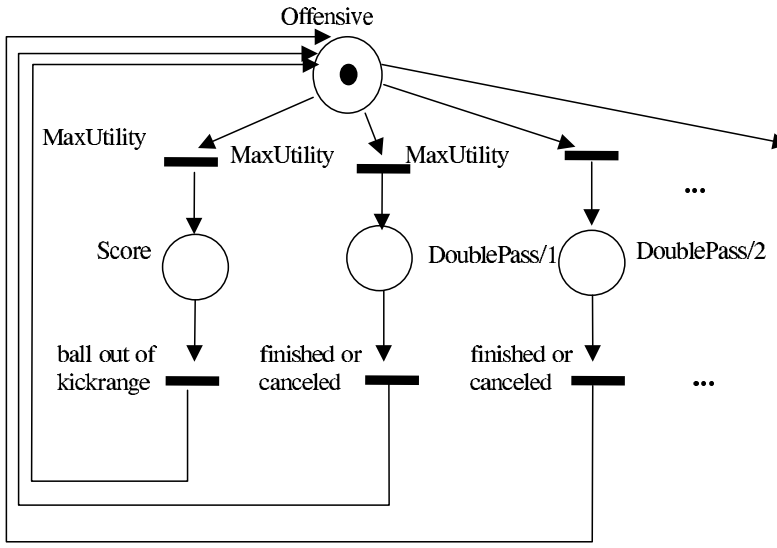


Fig. 4. Example of a Choice-Option

- Choice-Options can be performed by different, alternative subintentions (e.g. a pass can be performed by a forward-kick, a side ward-kick etc.), cf. Figure 4 for a Petri Net description of the alternatives of an "offensive option".
- Sequencing-Options are performed by a sequence of subintentions (e.g. the subintentions of a double pass as described in Figure 5 from the first player's perspective).

For clarity, the different kinds of connections are not mixed. This is similar to Prolog concepts: alternative subintentions correspond to different clauses of a predicate, sequenced subintentions to the subgoals in a clause.

Choice-options describe the different possibilities in the context of that option. Commitment and planning activities consist of choices from the alternative suboptions (e.g. using utilities), calculating appropriate parameters (e.g. for the kick direction) and decisions concerning the termination (or cancellation) of plans. Alternative plans can be provided. The hierarchical structure allows for local decisions. Redeliberation (if needed) can be performed in a given context.

Sequencing options describe the steps (suboptions) needed to perform a higher level option. There have to be well-defined criteria for the transitions from one suboption to the next one. The evaluation of these criteria is time critical because they should be decided immediately by the executor before acting in response to the newest sensory data.

According to deliberation and execution, options can be in different states. The deliberator chooses options to be executed. As long as they are prepared for future use, they are in the state "desired" and are called *desire*. As soon as they are under execution, they are in the state "intended" and are called

intentions. Intentions (and desires) form subtrees of the option-tree as shown for the double pass in Figure 3. The complete intention subtree must contain one subintention for each choice-option starting in the root down to some leafs, and all subintentions for each sequencing option. The intention tree has the form of a hierarchical partial plan. Subintentions describe the plan parts on different levels.

At any actual point in time, there exists a unique path in the intention subtree from the root to a leaf consisting of the *active* subintentions. It is called *activation path*. At the time when the first player passes to the second one, the activation pass in Figure 3 consists of "PlaySoccer"–"Offensive"–"DoublePass/1"–"Pass"–... down to a concrete action.

The executor performs the transitions from active subintentions to their sequential successors (which then become active) on all levels if related conditions are fulfilled. Using the least commitment principle, it computes parameters according to the newest sensory data. Moreover, the executor may stop the recent intention and switch to an alternative desire (which then becomes the intention). This appears if the situation does not proceed according to the plan. According to total dynamics, this can be initiated on any level.

The deliberator can prepare several desires as candidates for intentions. They serve as fast available alternatives for the executor when he has to stop a plan according to unexpected situations. As an example we might think about the fast switch to scoring a goal (because the situation allows it) instead of continuing the double pass.

7.2 Deliberator

The idea behind the deliberator is the preparation of intentions as partial hierarchical plans (built from options) without any hard time constraints (cf. Figure 3). It prepares this plan (as a desire) while the robot still performs an old intention. For example, while running for the ball the player may already evaluate its plans after intercept. At the same time, other players can evaluate their contributions to the possible plans of her team mate.

Standard situations provide generic cases of cooperative play. Using methods from CBR, a concrete situation can be matched to the standard situations. For

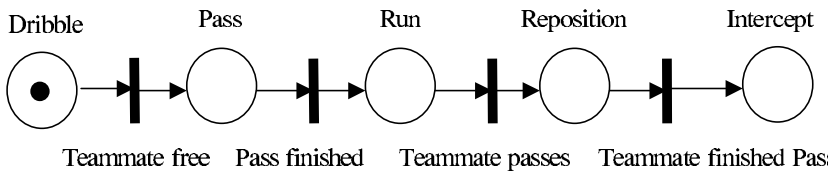


Fig. 5. Example of a Sequencing Option

example, a triggering feature for the double pass is an opponent attacking an offensive player controlling the ball. The standard situation (the "case") provides a standard strategy ("solution") for an intention. Using CBR methods for adaptation, a concrete intention is specified. The option hierarchy serves as a structure for describing cases.

7.3 Executor

Short time behavior in dynamic environments has to regard the newest available data, there is no place for time consuming deliberations. The executor works according to the activity path in the intention subtree. It starts from the root and proceeds level by level down to the recent leaf which specifies the next action to be executed. On each level it performs certain tests and calculates parameters according to the latest data (e.g. for performing an optimal kick). In the case of termination, it performs the transition to the next subintention. In exceptional situations, it may switch to a desire and make it to the new intention. The limited scope of decisions (embedded in the context of the intention) allows for fast computations.

The executor works as soon as new actions are to be performed, and as late as the latest data relevant for these actions can be analyzed. This can be done concurrently to the work of the deliberator - which at the same time prepares and specifies later activities for the executor. In a strictly sequential approach, the executor must interrupt the interpreter if necessary.

7.4 Main Features of the Double Pass Architecture

The Double Pass Architecture is organized as a doubled "one-pass-architecture": One-pass-architectures are described by a control flow which passes through each level only once. In our case, the control flow is directed from the highest level to the lowest one.

In the Double Pass Architecture, we have two separated passes: one pass for the deliberator which prepares intentions, and another pass for the executor which allows for quick reactions on all levels. The executor allows for a stimulus-response behavior on all levels, where the responses have been laid out by the deliberator. The executor realizes real-time behavior, while the deliberator can act without short time constraints.

The requirement to run through all levels by the executor needs a special runtime organization. Most runtime organization methods in programming are based on stacks, where a higher level method is called again only when the lower level has terminated. This holds for imperative languages as well as for descriptive ones. Existing implementations of BDI approaches like dMARS [dMARS] and JACK [JACK] use event queues for messages (external and internal) and intention-stacks for the scheduled (sub-)intentions. Hence the runtime behavior is similar to procedure calls. The run time behavior of hybrid architectures is similar, too. The reactive layers work with higher frequencies to act in real-time.

The deliberative layers have lower frequencies in the case of complex decision processes.

We leave the complex decision processes to the deliberator as far as they cannot guarantee real-time behavior. The necessary real-time behavior on the higher level is achieved using a limited scope of decisions according to the activation path in the intention subtree. This is performed by the executor. It can be understood as an implementation of bounded rationality in the sense of the BDI approach [Bratman87].

Moreover, as discussed in [Bratman87], the intentions are used to maintain stability to support coordination. It is maintained by high level behavior and well designed conditions for leaving an intention by the executor only in exceptional situations.

8 Further Work and Conclusion

Our project "Architectures and Learning on the Base of Mental Models" in the main research program 1125 "Cooperating teams of mobile robots in dynamic and competitive environments" of the German Research Association (DFG) investigates the usage of CBR methods for control of robots. The cases correspond to generic behavior which can be specified and adapted according to the current situation. There are two main goals for using CBR: The first goal is efficient control, while the second goal is learning from experience. Learning can be twofold: New cases can be acquired as templates for behavior, and the usage of existing cases can be improved by better analysis and adaptation methods.

The Double Pass Architecture has been implemented for the application of CBR methods. The option hierarchy is the structure for describing higher level cases. Analysis of situations concerns matching with cases in this case base. Adaptation is performed by least commitment strategies in the executor. Further development concerns completing the case base, refinements of analysis and adaptation, and usage for learning.

The main focus of this paper was the discussion of the real-time aspects of control in RoboCup. The Double Pass Architecture avoids some difficulties of layered architectures, it allows for real-time adaptations to new situations even on the higher levels and uses ideas of bounded rationality.

References

Arkin1998. R.C. Arkin: Behavior Based Robotics. MIT Press,1998.

Bratman87. M.E. Bratman. *Intentions, Plans, and Practical Reason.* Harvard University Press, Massachusetts, 1987.

Brooks91. R.A. Brooks: Intelligence without reason. Proceedings IJCAI-91, 569-595.

Burkhard00. H.D. Burkhard: Software-Agenten. In: G.Görz, C.-R.Rollinger, and J.Schneeberger: Einführung in die Künstliche Intelligenz. Oldenbourg 2000, 941-1015.

Burkhard *et al.*, 1998. Burkhard, H.D., Hannebauer, M. and Wendler, J.: Belief-Desire-Intention Deliberation in Artificial Soccer. *AI Magazine* 19(3): 87–93. 1998.

Burkhard2001. H.D. Burkhard: Real Time Control for Autonomous Mobile Robots. To appear in Fundamenta informaticae.

dMARS. dMARS: Technical Overview - 25 JUN 1996. `http://www.aaii.oz.au/proj/dmars_tech_overview/dMARS-1.html`

Dudek00. G. Dudek and M. Jenkin: Computational Principles of Mobile Robots. Cambridge University Press 2000.

Georgeff-Lansky87. M.P. Georgeff and A.L.Lansky: Reactive reasoning and planning. Proc. AAAI-87, 677–682, 1987.

JACK. `http://www.agent-software.com.au`

Kitano-et-al-97. H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, E. Osawa, and H. Matsubara. RoboCup: A challenge problem for ai. *AI Magazine*, 18(1):73–85, 1997.

Kolodner. J. Kolodner: Case Based Reasoning. Morgan Kaufman, San Mateo, CA, 1993.

Lenz-et-al98. M. Lenz, B. Bartsch-Spörl, H.D. Burkhard, S. and Wess (Eds.): Case Based Reasoning Technology. From Foundations to Applications. LNAI 1400, Springer 1998.

Maes90. P. Maes (Hrsg.): Designing Autonomous Agents. Theory and Practice from Biology to Engineering and Back. MIT Press 1990

JPMüller96. J.P. Müller: The Design of Autonomous Agents – A Layered Approach. LNAI 1177, 1996.

Müller-Gugenberger and Wendler, 1998. P. Müller-Gugenberger and J. Wendler: *AT Humboldt 98 — Design, Implementierung und Evaluierung eines Multiagentensystems für den RoboCup'98 mittels einer BDI-Architektur*. Diploma Thesis. Humboldt University Berlin, 1998.

Murphy00. Robin R. Murphy: Introduction to AI. MIT Press, 2000.

Parunak97. Van Parunak: 'Go to the Ant': Engineering Principles from Natural Agent Systems. Annals of Operations Research, 1997.

RaoGeorgeff91. A.S. Rao and M.P. Georgeff. Modeling agents within a BDI-architecture. In R. Fikes and E. Sandewall, editors, *Proc. of the 2rd Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'91)*, 1991.

RoboCup. RoboCup. The Robot World Cup Initiative: `http://www.robocup.org`. Proceedings of the RoboCup Workshops/Symposia in the Springer LNAI-Series.

RussellNorvig95. S. Russell and P. Norvig: Artificial Intelligence: A Modern Approach. Prentice-Hall, 1995.

Wendler-et-al00. J. Wendler, S. Brüggert, H.D. Burkhard, and H. Myritz: Fault-tolerant Self Location by Case Based Reasoning. In: Peter Stone, Tucker Balch, and Gerhard Kraetzschmar (Eds.): RoboCup 2000: Robot Soccer World Cup IV. Springer, LNAI 2019, 259–268.

WendlerLenz98. J.Wendler and M.Lenz: CBR for Dynamic Situation Assessment in an Agent-Oriented Setting. In D. Aha and J. Daniels (eds.): Proc. of the AAAI'98 Workshop on Case-Based Reasoning Integrations,1998, Madison, USA.

Perceptual Anchoring: A Key Concept for Plan Execution in Embedded Systems

Silvia Coradeschi and Alessandro Saffiotti

Center for Applied Autonomous Sensor Systems
Dept. of Technology, Örebro University
S-70182 Örebro, Sweden
{silvia.coradeschi,alessandro.saffiotti}@aass.oru.se
http://www.aass.oru.se

Abstract. Anchoring is the process of creating and maintaining the correspondence between symbols and percepts that refer to the same physical objects. This process must necessarily be present in any physically embedded system that includes a symbolic component, for instance, in an autonomous robot that uses a planner to generate strategic decisions. However, no systematic study of anchoring as a problem *per se* has been reported in the literature on intelligent systems. In this paper, we advocate for the need for a domain-independent framework to deal with the anchoring problem, and we report some initial steps in this direction. We illustrate our arguments and framework by showing experiments performed on a real mobile robot.

1 Perceptual Anchoring

It's ten o'clock and I need more coffee. I tell Milou, my personal robot, to go and fetch my cup of coffee, which is on the kitchen table. Milou rolls to the kitchen, approaches the table, and uses its camera to find an object that looks like a cup of coffee. Two cups are standing on the table, one filled with chocolate and one with coffee. From the camera image, both cups match the given description "cup of coffee," so Milou decides to acquire more information. Milou is equipped with an electronic nose that can be used to identify coffee by smell. It turns toward the first cup, approaches it, and smells it: this does not smell like coffee. It then turns to recover sight of the second cup, approaches it, and smells it: this time the odor matches the intended signature. The camera image gives enough information to accurately estimate the position of the cup, so Milou can comfortably grasp it and bring it to my room.

This hypothetical scenario illustrates a common mechanism of our everyday life: the use of words to refer to an object in the physical world, and to communicate this reference to another agent. In this case, "my cup of coffee which is on

M. Beetz et al. (Eds.): Plan-Based Control of Robotic Agents, LNAI 2466, pp. 89–105, 2002.

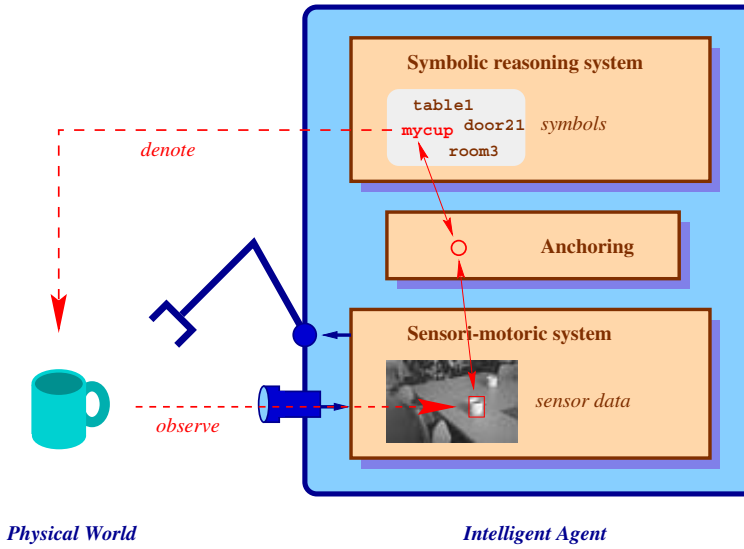


Fig. 1. Graphical illustration of the anchoring problem

the kitchen table". Our ability to deploy intelligent robots that can provide services to non-technical human users critically depends on our ability to develop mechanisms like this for human-robot interaction. An important prerequisite for this is that the robot must be able to establish the right correspondence between the *symbols* used to refer to a given object, and the *perceptual data* acquired by observing that object. We call **anchoring** the process of creating and maintaining the correspondence between symbols and sensor data that refer to the same physical objects — see Figure 1.

A closer look to anchoring reveals that this is a complex process, which requires the use of several functionalities. In our initial example, Milou is given a linguistic (symbolic) description of the object named "my cup", and it must *find* the intended object through perception. To do so, it must match the data coming from its sensors against this description; it must also recognize ambiguities and decide how to resolve them, e.g., by acquiring more information. Once the object has been identified, the robot must *track* it while moving. Finally, the object may temporarily disappear from the sensor's view, e.g., because it has moved, or because the gaze is turned to another direction. The robot must maintain a virtual image of the object in memory, and be able to *reacquire* the object when it comes back into view. There is currently no general theory that tells us how to define and combine these functionalities to perform anchoring. In this paper, we advocate the need of such a theory, and outline our initial steps toward its development.

2 Why Plan-Based Robots Need Perceptual Anchoring

Since its conception in 1956, the field of AI has been pursuing the objective of building intelligent agents that can move and operate in the physical world. For many years, however, the belief that the interesting issues were at the level of the abstract reasoning processes, together with the disconcerting difficulties of perception, kept most AI researchers isolated from embedded systems. Typical AI systems, like expert systems, did not try to directly sense or act upon the physical world: humans did the job of translating observations of the world into the symbols used by these systems, and translating those symbols back to actions in the world. It is only recently that intelligent systems started to be more and more often connected with sensors and actuators to produce *embedded intelligent systems* able to perform useful operations in real world environments (e.g., [3,11,21]).

An embedded intelligent system must incorporate motor and perceptual processes to interface with the physical world, and abstract cognitive processes to reason about the world and the available options. In many cases, the abstract processes are symbol-based: when they rely on classical AI techniques, but also when we want our robots to use linguistic communication to interact with humans. In these cases, a crucial aspect of the integration between the cognitive and sensori-motoric level is the connection between the *symbols* used by the symbol system to denote a physical object in the world, and the *sensor data* in the sensori-motoric system that corresponds to the same object (see Fig. 1). The problem of how to create this connection, and how to maintain it in time, is exactly the anchoring problem.

Autonomous robots that incorporate a symbolic planner to plan their operation are examples of intelligent embedded systems. In these robots, anchoring is needed in order to associate the terms used in the plan to the relevant sensor data. Consider for instance a plan that contains the action "PickUp(cup-22)," where "cup-22" is the symbol used by the planner to denote the specific object (say, Alex' cup) to be used to achieve the given goal (say, bring Alex some coffee). In order to execute this action, the robot has to: (i) identify the sensor data in the perceptual stream that pertains to that object; and (ii) use these data in a sensori-motor loop to execute the grasping on the correct object. That is, it has to anchor the symbol "cup-22" to the perceptual data corresponding to the intended object.

Although anchoring must necessarily take place in any robotic system that comprises a symbolic planning and reasoning component, an analysis of the existing literature reveals that the anchoring problem has received little attention in the fields of AI and autonomous robotics as a problem *per se* (see Section 5.1). Instead, anchoring is typically solved on a system-by-system basis on a restricted domain, and the solution is hidden in the code. This is unfortunate, since a deep study of the anchoring problem would allow us to develop a set of common principles and techniques that can be applied to any such system. In a more general perspective, a study of anchoring would increase our understanding of

the delicate issue of integration between symbolic planning and reasoning on the one hand, and physical perception and action on the other hand.

To the best of our knowledge, the first domain independent definition of the anchoring problem was given in [24], while the first attempt at defining a computational theory of anchoring was reported in [7]. In what follows, we outline the basic elements of this theory, and show its relevance to the design of plan-based autonomous robots by discussing a few examples.

3 A Formal Model of Anchoring

We give here an outline of the formal model of perceptual anchoring proposed in [7] and [9]. The reader is addressed to those references for more details on the model.

3.1 The Ingredients of Anchoring

We consider an intelligent embedded system that includes the following elements.

- A *symbol system* Σ, which contains individual symbols (variables and constants), predicate symbols, and an inference mechanism. Our interest is directed to the individual and predicate symbols. Examples of individual symbols are `mycup`, `suitcase1`, and `car2`; examples of predicates are `large`, `small`, and `red`.
- A *perceptual system* Π, which includes percepts and attributes. We take a percept to be a structured collection of measurements that are assumed to originate from the same physical object; an attribute is a measurable property of percepts. Examples of percepts are image regions identified as representing objects; common attributes computed on these percepts are *color*, *width*, and *area*.
- A *predicate grounding relation* g, which embodies the correspondence between unary predicates and values of measurable attributes. For instance, g may encode the correspondence between the predicate `red` and the corresponding Hue values measured in the image.[1] We do not make any assumption about the origin of g: for instance, g can be hand-coded by the designer of the system, or it can be learned by the system.

We are not concerned with the internal details of Σ, Π, and g here. These can be whatever, as long as they include the elements listed above. What interests us is how to connect these ingredients in order to perform anchoring.

Lets consider a simple example to illustrate the ingredients of anchoring. Σ may be a planner that includes the individual symbol 'A' and the predicate symbols 'large' and 'small.' Ξ may be a vision system able to recognize suitcases:

[1] The use of crisp definitions for colors is obviously problematic, and more complex forms of g may be needed in practice. In our implementation, for example, we use fuzzy logic [6] and [5]. We assume a crisp g here for sake of simplicity.

from the image shown in Fig. 2, Ξ may extract two percepts π_1 and π_2. Attributes computed by Ξ may include 'color' and 'width.' The predicate grounding relation g may include the triple $\langle \text{small}, \text{width}, 10 \rangle$: this says that the measure 10 for an object's observed width is consistent with the predication of its being small.[2]

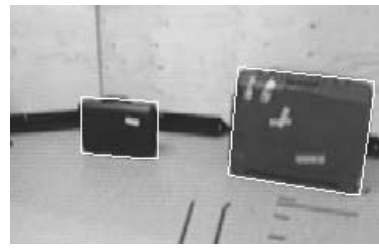


Fig. 2. Two percepts extracted from a camera image

At every point in time, Σ contains a set of individual symbols and Π contains a set of percepts. Moreover, both systems contain information about the properties associated to these symbols and percepts. This information is represented as follows.

- The *symbolic description* σ of a symbol is a set of predicates[3] that are predicated in Σ of the symbol; an example of a symbolic description is {`small`, `red`},
- The *perceptual signature* γ of a percept is a function that gives the values of all the attributes of the percept, as measured by Π. Γ is the set of all signatures. An example of perceptual signature is: $\gamma(\text{width}) = 230, \gamma(\text{area}) = 380$.

Let us consider our previous example. At time t, the symbol system may associate property 'small' to symbol 'A' by having `small` belonging to the symbolic description of 'A'. The perceptual system may extract the width of the two percepts π_1 and π_2 in the image, and associate them with their respective perceptual signatures γ_1 and γ_2 such that $\gamma_1(\text{width}) = 10$ and $\gamma_2(\text{width}) = 20$.

The task of anchoring is to use the above ingredients to create and maintain the right correspondence between symbols in Σ and percepts in Π. The pivot to this correspondence is the matching between a symbolic description and a perceptual signature: intuitively, we connect a percept and a symbol if their properties are compatible according to g. This correspondence is reified in an internal data structure, called an *anchor*, that uniquely represents a specific

[2] For the sake of simplicity we consider here a very simple g. The g relation can be quite complex in real domains.

[3] We currently consider in our framework just unary predicates, meant to denote properties of individual objects. The extension to n-ary predicates, meant to denote relations among individuals, is part of our future work.

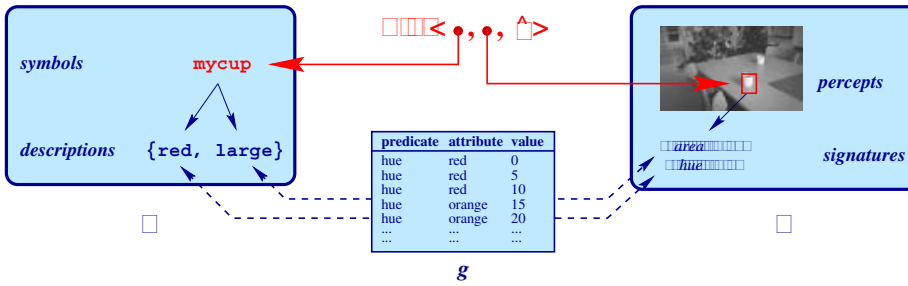


Fig. 3. The elements of our model of anchoring. The task of anchoring is to create and maintain the anchor α for a given object

physical object. The anchor contains three elements: the *symbol* that denotes that object inside Σ; the *percept* from Π that has been matched to that symbol; and a *perceptual signature* $\hat{\gamma}$ that gives the current estimate of the observable properties of the object.

Definition 1. *An* anchor α *is any partial function from time to triples in* $\mathcal{X} \times \Pi \times \Gamma$.

The observable properties of an object can be used to *act* on the object (e.g., the observed position is needed in order to approach the object), or to *reidentify* the object at a later time. The elements described above are illustrated in Figure 3.

The task of the anchoring process is to create and maintain an anchor for each specific object of interest using the elements $\langle \Sigma, \Pi, g, \sigma, \gamma \rangle$ as basic ingredients.

3.2 The Dynamical Aspect

As we will shortly see, the initial creation of an anchor resembles a structural patter recognition process. Once an anchor has been created, however, this must be continuously updated to account for changes in the symbolic properties, the acquisition of new percepts in the perception stream, or the change of properties with time. This is especially important in a dynamic environment where objects move around. In general, updating is based on a combination of prediction and new observations, as illustrated in Figure 4. Prediction is used in order to make sure that the new percepts used in re-anchoring a symbol are compatible with the previous observations. In other words, we want to make sure that we are still tracking the same object. Comparison with the symbolic descriptor is used to make sure that the updated anchor still satisfies the predicated properties. In other words, we want to be sure that the object still has the properties that make it "the right one" for the goals of the symbolic system.

The main outcome of the update is the computation of new signature, which is stored in the anchor. At every time t, this signature provides an estimate of the

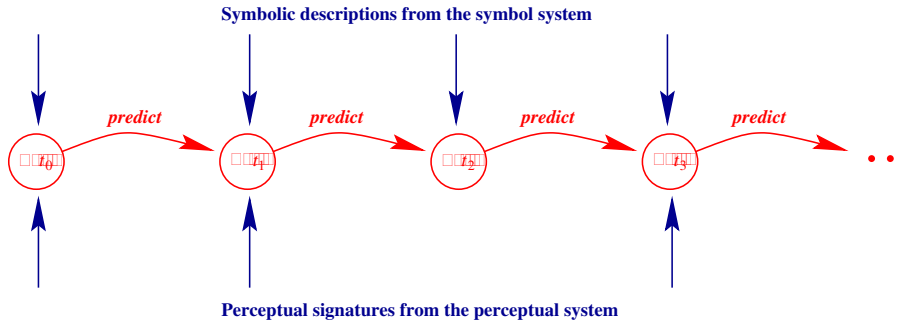


Fig. 4. Anchor dynamics. After its initial creation, the anchor is constantly updated by a combination of prediction and new observations

observable properties of the object; these may be used, for instance, by a control system to guide action. When no matching percept is found, the signature stored in the anchor is only based on prediction.

The anchor dynamics resembles the usual predict-measure-update cycle of recursive estimators, e.g., of a Kalman filter. There is, however, an important difference: in updating the anchor, we also consider the abstract information (symbolic descriptor) provided by the symbol system. The experiment shown in Section 4.2 below illustrates a case where this information is crucial to a correct anchoring.

3.3 The Functionalities of Anchoring

The anchoring process described above can be defined by three abstract functionalities:

Find. This functionality corresponds to the initial creation of an anchor for an object given its symbolic description. This functionality selects the adequate percept from the perceptual stream provided by the perceptual system Π according to a domain-specific matching function, which uses the g predicate grounding relation. If several matching percepts are found, then one of them is selected using a domain-specific selection function. (In [9] we consider the possibility of creating several anchors corresponding to several anchoring hypotheses.) This functionality is summarized in Figure 5.

Track. This functionality corresponds to dynamical update of the anchor to take into account the passage of time and the arrival of new percepts in the perceptual stream. This functionality relies on domain-dependent functions for matching and selecting, as above, plus functions for predicting and updating the anchor's signature. This functionality is summarized in Figure 6.

Reacquire. It is useful to distinguish the case where the object is kept under constant observation, which is solved by the Track functionality, from the

```
procedure Find (x)
   percept ← Select a percept such that its perceptual signature
             matches the symbolic description for x;
   if percept = null
     then fail
     else create an anchor storing the symbol x,
          the percept, and its perceptual signature
   return anchor
```

Fig. 5. Algorithm for a general FIND functionality

case where the object is re-observed after some time. The Reacquire functionality takes care of this case. Although the general algorithm for Reacquire resembles the one for Track, the domain-dependent functions may be different. For instance, the Predict function may involve more complex reasoning about how the observed properties may have changed, and which ones of them should still be considered in the match.

Reacquire is somehow a combination of the Find and the Track functionalities. For example, consider a robot that has executed the action "PutDown(cup-22)," has gone to attend to another task, and needs later on to perform the action "PickUp(cup-22)." The robot needs to re-establish the anchor for "cup-22." This should not be achieved simply by a Find functionality, since the robot has some perceptual information about the cup (e.g., its shape and color as it was perceived, or its position) which may be more detailed than the symbolic descriptor for it. This cannot be achieved simply by the Track functionality either, since the prediction may be more complex (e.g., the lighting conditions may be different, thus making the previous color measurement difficult to use). Part of the prediction can be done inside the symbolic system Σ, e.g., by hypothetical reasoning. The result of the symbolic-level prediction would then be an updated symbolic description, which would then be fed into the Reacquire functionality.

In our experiments, we have found that FIND, TRACK, and REACQUIRE form a complete set of functionalities that is sufficient to solve the anchoring problem in all the cases that we have considered until now.

4 Using the Model

Does our formal model contain all the basic elements which are needed in general to perform anchoring? In order to start answering this question, we have tried to use this model to solve the anchoring problem in several different autonomous robots, equipped with different symbolic planners and different perceptual components, and we have tested them in different domains.

procedure Track (x)
  anchor ← current anchor for x
  signature ← Predict signature at current time from *anchor*
  percept ← Select a percept such that its perceptual signature matches the symbolic description of x and the predicted *signature*
  if *percept* = null
    then Update *anchor* with *signature*
    else Update *anchor* by combining the predicted *signature* and the perceptual signature of *percept*
  return *anchor*

Fig. 6. Algorithm for a general TRACK functionality

4.1 A Robot Navigation Experiment

One such experiment was performed on a Nomad 200 mobile robot. The robot was controlled by a system comprising a symbolic module, consisting in a world model and a conditional planner, [18] and [19], a perceptual module using vision, and the Thinking Cap navigation system.[4] The architecture of the system is shown in Figure 7. The Symbol and perceptual systems are outlined. The task was to approach a small black suitcase with a white label. (The full task would involve fetching the suitcase, but our robot currently does not have a manipulator.) The initial set-up is shown in Figure 8 (left).

To perform this task, the planner is given the goal to be near a small black suitcase with a white mark:

```
(exists (?x)
   (and (suitcase ?x) (small ?x) (black ?x) (white-label ?x) (near ?x)))
```

where `?x` is a variable. The world model contains information about three suitcases, identified by the symbols `A`, `B`, and `C`: `A` is a large green suitcase, while `B` and `C` are small black ones. However, the world model does not contain any information about labels. Therefore, the planner generates a conditional plan:

```
((gonear C) (observe C)
 (if ((white-mark C . true)) (:success))
 (if ((white-mark C . false))
     ((gonear B) (observe B)
      (if ((white-mark B . true)) (:success))
      (if ((white-mark B . false)) (:fail)))))
```

What this plan says is the following: First go near suitcase `C` and look for a white label; if this is found, then we are done; otherwise, go near suitcase `B` and look for a white label; if this is found, then we are done; otherwise, we have failed. The reason why the robot needs to go near a suitcase before performing an `Observe` action is that the planner knows that labels can only be perceived from close by.

[4] The Thinking Cap is an autonomous robot architecture based on fuzzy logic [23]. This is a successor of the architecture originally developed for the robot Flakey [25].

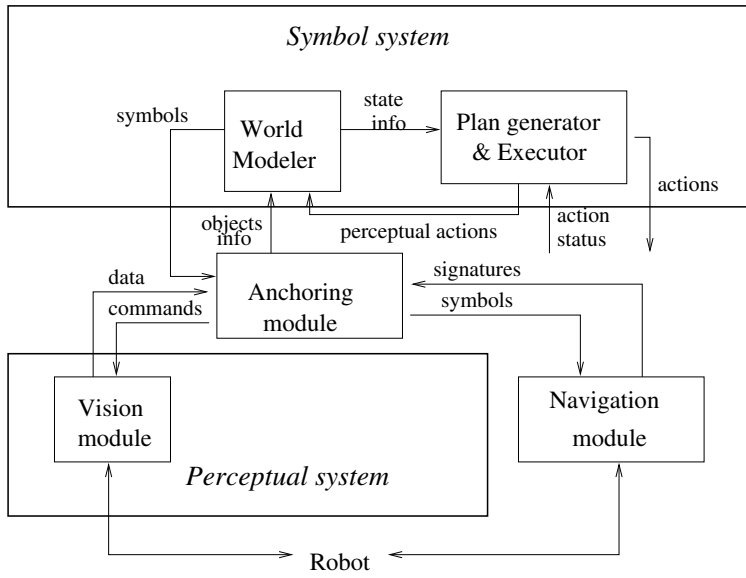


Fig. 7. System architecture in the robot navigation experiment

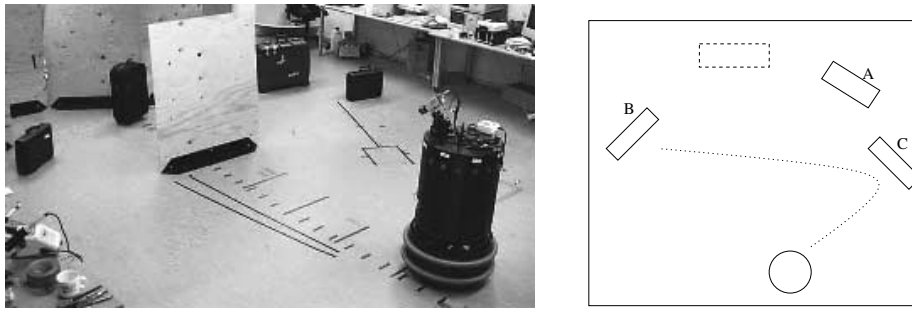


Fig. 8. Anchoring in action. To execute the action `(gonear C)` generated by the planner, the robot must anchor the symbol `C` to the correct suitcase in the environment

After the plan has been generated, it must then be executed: here is where the anchoring problem arises. To execute the first action `(gonear C)`, the symbol `C` must be given a meaning from the point of view of the sensori-motoric system, that is, it must be anchored to specific sensor data using the Find functionality. These data are then used by the navigation system to direct the robot toward the correct physical object as perceived by the vision system.

Two types of information are available to the Find functionality: (1) the symbolic description of `C` provided by the world model in the symbol system: {`suitcase, small, black`}; and (2) the objects identified by the vision system

(percepts), together with their perceptual signatures. In our case, the vision system identifies three percepts as suitcases and measures their size, color, and position, together with the presence and color of a label. To create an anchor for `C`, the Find selects a percept whose signature best matches the symbolic description of `C` based on the predicate grounding relation g.

Once the best matching percept is found, the anchor is created, and is is filled with the observed properties of the object, as measured on the percept. These properties are used to perform action: for instance, the position of the suitcase is used by the navigation system to perform the "gonear" action. During navigation, the suitcase is constantly tracked using the Track functionality, and the properties of the anchor (e.g., its position relative to the robot) are constantly updated.

In our experiment, the robot navigated to suitcase `C` but did not find any white label on it, so it had to execute the `(gonear B)` action. To do so, it turned toward the expected position of `B`, as stored in the world model, anchored `B` to the perceived suitcase, and used the observed position to precisely navigate in front of it. From there, the robot could actually observe a white label on the suitcase, so the task completed successfully, as shown in Figure 8 (right).

4.2 A UAV Experiment

The following experiment stresses the dynamical aspect of the anchoring problem. This experiment, performed in the framework of the WITAS project [11], involves an autonomous helicopter performing traffic surveillance tasks in a simulated environment. The helicopter is controlled by a system that integrates a

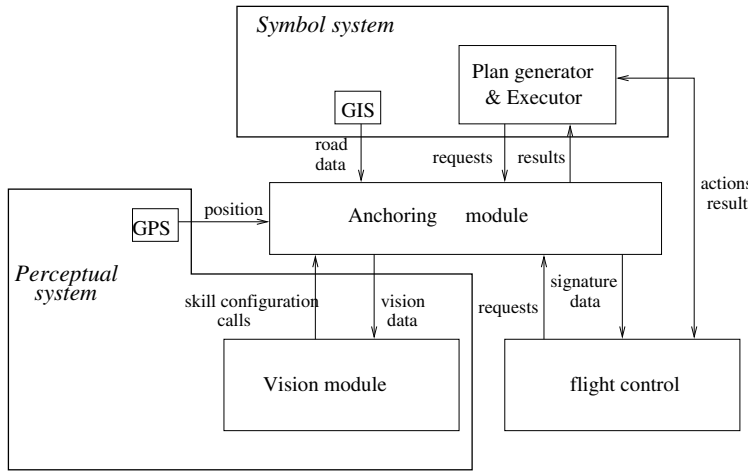
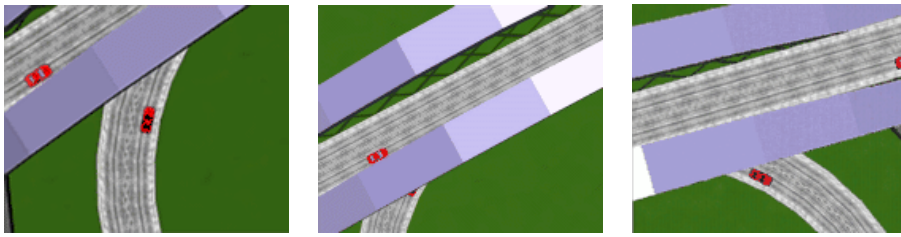


Fig. 9. Simplified view of the architecture of the WITAS system as it was when the experiment was performed (December 1999)



(a) (b) (c)

Fig. 10. A difficult case of anchor maintenance. The followed car disappears under a bridge and a similar car appears over the bridge

planner, a reactive plan executor, a vision system, and a flight control system. An anchoring modules connects the plan executor to the vision and the control systems. Figure 9 shows a simplified view of the architecture of the WITAS system as it was when the experiment was performed (December 1999). The helicopter is following a car that had been previously anchored (FIND). The flight control system takes the current position of the car from the signature in the anchor. The TRACK functionality is used to regularly update this anchor using the percepts provided by the vision system. In this experiment, we mainly relied on a Kalman filter for this functionality.

Where the experiment becomes more interesting is in the case shown in Figure 10. The helicopter is following the car in the middle of the image (a), which has already been anchored. At (b), our car disappears under the bridge. Simultaneously, an identical car appears on the bridge at the position in the image where the first car would have been, had it not been occluded by the bridge. Now, the TRACK routine predicts the perceptual signature of the first car's anchor. In particular, its expected position is extrapolated from its previous position and speed, and from higher level information derived from the symbol system concerning the road on which the car was traveling and the topology of the road network. The percept provided by the vision system is the image region containing the second car over the bridge. The attributes of this percept are compared to the predicted perceptual signature. Using high-level information, the percept is discarded: the intended object must be on the road passing under the bridge and not on the bridge — and we know that cars can only change roads at a crossing. Thus, the anchor of the first car is updated by prediction only, until an appropriate percept is found.

The flight controller can still use the predicted position in the anchor to maneuver the helicopter. However, because the continuous tracking has failed, the anchoring system now goes into the REACQUIRE modality. This modality uses a more complex prediction model, which also includes information about the road network and about possible car behaviors. This makes the predicted position of the car to be at the other end of the bridge. This prediction, stored in the anchor, is used to maneuver the helicopter and the camera accordingly. When eventually the first car reappears from under the bridge (c), a percept

is generated which is compatible with the predicted signature, and the anchor is updated using this percept. We emphasize that the key to perform a correct anchoring in this example is the combined use of previously perceived attributes and symbolic domain knowledge in both prediction and matching.

5 Discussion

5.1 Work Related to Anchoring

Although anchoring as defined in this paper has not been the subject of previous rigorous investigation, issues related to anchoring have been discussed in the fields of autonomous robotics, machine vision, linguistics, and philosophy.

The autonomous robotics literature contains a few examples in which the need and the role of anchoring, under different names, has been explicitly identified (e.g., [15,25]). Jung and Zelinsky [17] use a similar concept to achieve grounded communication between robots. None of these works, however, pursue a systematic study of the anchoring problem. Bajcsy and Košecká [1] offer a general discussion of the links between symbols and signals in mobile robotic systems. Yet, they do not deal with the problem of how to create and maintain these links, which is the main issue in anchoring.

The machine vision community has done much work on the problems of object recognition and tracking. While anchoring relies on these underlying perceptual abilities, it is mainly concerned with the integration of these with a symbol system. Some work in vision has explicitly considered the integration with symbols. Satoh *et al.* [27] present a system which associates faces and names in news videos looking at co-occurrences between the speech and the video streams. Horswill's Ludwig system [16] answers natural language queries by associating linguistic symbols to markers and to marker operations in the image space. Interestingly, Ludwig may refer to physical objects using indexical terms, like "the block on the red block." Markers are also used by Wasson *et al.* [29] to provide a robot with a perceptual memory similar to our anchors above. All the work, however, describe specific implementations, and they do not attempt a study of the general anchoring concept.

The problem of connecting linguistic descriptions of objects to their physical referents has been largely studied in the philosophical and linguistic tradition, most notably in the work by Frege and by Russell [12,22]. In fact, we have borrowed the term *anchor* from situation semantics [2], where this term denotes an assignment of variables to individuals, relations, and locations. These traditions provide a rich source of inspiration for the conceptualization of the anchoring problem, but they typically disregard the formal and computational aspects necessary to turn these ideas into techniques.

Anchoring is related to two large research areas: pattern recognition and symbol grounding. Pattern recognition is the problem of how to recognize a pattern given sensory measurements. Symbol grounding [14] is the problem of how to give an interpretation to a formal symbol system that is based on something

that, contrary to classical formal semantics, is not just another symbol system. Anchoring can be seen an important, concrete aspect that lays in the intersection of these research areas. It shares with symbol grounding the assumption that a symbolic system is present, which is not necessary the case in Pattern recognition; and it shares with pattern recognition the assumption that sensory measurements are used, which is not necessary the case in Symbol grounding. Moreover, anchoring focus on a specific problem: connecting symbol-level representations of individual objects to the perceptual image of these objects. While symbol grounding and pattern recognition are very wide problems for which there is little hope to find a "general" solution, anchoring is a more restricted problem for which we can hope to find a practical and general solution.

A peculiar aspect of anchoring is its reliance on internal representations that are uniquely associated to each object of interest, and which include all the sensor-level properties needed to operate on these objects. In our proposal, these representations are provided by the anchors, but other systems that perform anchoring also incorporate similar representations — see, for instance, the markers used in [16] or the PML-structures in [28]. Anchors provide perceptual handles to the actual objects denoted by symbols, so that we can perform physical actions on them. Anchors also provide a means to share a reference to a physical object between different sub-systems. In our examples, we have seen the use of anchors to share this reference between a symbolic planner, a vision system, and a motor control system.

5.2 Where to Go Next

The above treatment of anchoring must be seen as a starting point. Anchoring hides a number problems that need to be carefully investigated. Some of these problems are technical in nature, and their solution is needed in order to apply any theory of anchoring to complex domain. For instance, perceptual information is typically affected by *uncertainty*, and this uncertainty should be taken into account in the anchoring process. Moreover, the predicates used in symbolic descriptions can be inherently *vague*. This is especially true for many predicates commonly used in linguistic human-robot communication, like "red" or "large." In our actual implementations, we have accounted for both these factors using fuzzy logic [6] and [5], but other solutions may be possible. Furthermore, the meaning of many predicates in terms of physical quantities is highly context dependent, as in the case of the predicate "red" when referred to wine. An interesting possibility to address some of these problems would be to frame the predicate grounding problem in the context of Gärdenförs' conceptual spaces [13]and [4]. Finally, in cases of perceptual ambiguity we may need to maintain *multiple hypotheses* for the anchor. Multiple hypotheses are also needed in the presence of partial matching, that is, when some of the attributes corresponding to the required predicated cannot be observed [9].

Other problems are more conceptual, and they are related to some subtle issues in the definition of anchoring. One such issue is the distinction between

definite descriptions, like "the cup of coffee on the table," and *indefinite* descriptions, like "a cup of coffee." These descriptions need different treatments, e.g., if the robot sees two cups on the table. A second important issue is the origin of the *g predicate grounding relation*. In our example, this relation was built-in by the system designer, but this may not be adequate in other applications. For instance, in the case of the artificial nose mentioned in our opening scenario, there is no well-established numerical model of the sensor. Therefore, the relation between predicates denoting properties that relate to smell and the actual measurements acquired from this sensor has to be learned rather than given a priori. A preliminary experiment in this sense is reported in [20], where an artificial neural network has been trained to recognize vanilla-, lavender-, and yogurt-like aroma using an artificial nose. Finally, the information contained in the anchor can be used to reason on the use of *perceptual resources*. In both experiments reported above we have used the expected properties of the object, stored in the anchor's signature, to direct the video camera and to parameterize the vision routines. In another application [26], we have used this information to select the focus of attention when tracking multiple objects.

6 Conclusions

The problem of perceptual anchoring can be extraordinary complex, and many of its subtleties have been the object of much thought throughout the history of philosophy. Nonetheless, if we want to build a physically embedded agent that incorporates a symbolic component, we have to solve (a specific instance of) it. This is true in particular for Plan-based robotic systems. In this paper, we have advocated the study of a theory of anchoring that is general enough to allow solutions to be ported across different systems, but still specific enough to be manageable.

The quest for such a theory is still in its initial phase, but it is already elicitating much interest [8] and [10]. It will probably have an inter-disciplinary flavor, combining insights from the study of symbol grounding, estimation theory, belief revision, pattern recognition, and still others. Inter-disciplinary will also be needed because a general theory of anchoring must be solidly grounded in experiments performed on many different systems operating in many different domains. We believe that having such a theory will greatly advance our ability to build intelligent embedded systems.

Acknowledgments

This work was funded by the Swedish KK Foundation. We thank Lars Karlsson and Zbigniew Wasik for providing substantial help in running the robot experiments.

References

1. R. Bajcsy and Košecká. The problem of signal and symbol integration: a study of cooperative mobile autonomous agent behaviors. In *Proceedings of KI-95: Advances in Artificial Intelligence, 19th Annual German Conference on Artificial Intelligence*, volume 981 of *LNCS*, pages 49–64, Berlin, Germany, 1994. Springer.
2. J. Barwise and J. Perry. *Situations and Attitudes*. The MIT Press, 1983.
3. W. Burgard, A. Cremers, D. Fox, D. Hähnel, G. Lakemeyer, D. Schulz, W. Steiner, and S. Thrun. Experiences with an interactive museum tour-guide robot. *Artificial Intelligence*, 114(1-2):3–55, 1999.
4. A. Chella, M. Frixione, and S. Gaglio. Anchoring symbols on conceptual spaces: the case of dynamic scenarios. *To appear in [10]*, 2003.
5. S. Coradeschi, D. Driankov, L. Karlsson, and A. Saffiotti. Fuzzy anchoring. In *In Proc. of the 10th IEEE International Conference on Fuzzy Systems*, Melbourne, December 2001.
6. S. Coradeschi and A. Saffiotti. Anchoring symbols to vision data by fuzzy logic. In *Qualitative and Quantitative Approaches to Reasoning with Uncertainty*, LNAI, pages 104–115. Springer, Berlin, Germany, 1999.
7. S. Coradeschi and A. Saffiotti. Anchoring symbols to sensor data: Preliminary report. In *Proc. of the 17th National Conference on AI (AAAI-2000)*, pages 129–135, 2000. Online at `http://www.aass.oru.se/~asaffio/`.
8. S. Coradeschi and A. Saffiotti, editors. *Anchoring Symbols to Sensor Data in Single and Multiple Robot Systems. Papers from the 2001 AAAI Fall Symposium*, Technical Report FS-01-01. AAAI, 2001.
9. S. Coradeschi and A. Saffiotti. Perceptual anchoring of symbols for action. In *Proc. of the 17th IJCAI Conf.*, pages 407–412, Seattle, WA, 2001. Online at `http://www.aass.oru.se/~asaffio/`.
10. S. Coradeschi and A. Saffiotti, editors. *Anchoring Symbols to Sensor Data in Single and Multiple Robot Systems. Special issue of the Robotics and Autonomous Systems journal.*, 2003.
11. P. Doherty, G. Granlund, K. Kuchcinski, E. Sandewall, K. Nordberg, E. Skarman, and J. Wiklund. The WITAS unmanned aerial vehicle project. In *Proc. of the European Conf. on AI (ECAI)*, Berlin, 2000.
12. F.L.G. Frege. Über Sinn und Bedeutung. *Zeitschrift für Philosophie und philosophische Kritik*, pages 25–50, 1892.
13. P. Gärdenförs. *Conceptual Spaces: The Geometry of Thought*. MIT Press, 2000.
14. S. Harnard. The symbol grounding problem. *Physica D*, 42:335–346, 1990.
15. H. Hexmoor, J. Lammens, and S. C. Shapiro. Embodiment in GLAIR: A grounded layered architecture with integrated reasoning for autonomous agents. In D. Dankel, editor, *Proceedings of the Florida AI Research Symposium*, pages 325–329, 1993.
16. I. Horswill. Visual architecture and cognitive architecture. *Journal of Experimental and Theoretical Artificial Intelligence*, 9(2):277–292, 1997.
17. D. Jung and A. Zelinsky. Grounded symbolic communication between heterogeneous cooperating robots. *Autonomous Robots*, 8(3):269–292, 2000.
18. L. Karlsson. Conditional progressive planning: A preliminary report. In *Proc. of the 17th IJCAI Conf.*, Seattle, WA, 2001.
19. Karlsson L. and T. Schiavinotto. Progressive planning for mobile robots — A progress report. In *This volume*. Springer, 2002.

20. A. Loutfi, S. Coradeschi, T. Duckett, and P. Wide. Odor source identification by grounding linguistic descriptions in an artificial nose. In *Proc. of the SPIE conference on Sensor Fusion*, Orlando, 2001.
21. N. Muscettola, P. Nayak, B. Pell, and B. Williams. Remote agent: To boldly go where no ai system has gone before. *Artificial Intelligence*, 103(1-2):5–48, 1988.
22. B. Russell. On denoting. In *Mind XIV*, pages 479–493. 1905.
23. A. Saffiotti. The Thinking Cap home. `http://www.aass.oru.se/~asaffio/Software/TC/`.
24. A. Saffiotti. Pick-up what? In C. Bäckström and E. Sandewall, editors, *Current trends in AI Planning*, pages 266–277. IOS Press, Amsterdam, Netherlands, 1994.
25. A. Saffiotti, K. Konolige, and E. H. Ruspini. A multivalued-logic approach to integrating planning and control. *Artificial Intelligence*, 76(1-2):481–526, 1995.
26. A. Saffiotti and K. LeBlanc. Active perceptual anchoring of robot behavior in a dynamic environment. In *IEEE Int. Conf. on Robotics and Automation*, pages 3796–3802, 2000.
27. S. Satoh, Y. Nakamura, and T. Kanade. Name it: Naming and detecting faces in video by integration of image and natural language processing. In *Proc. of the 15th IJCAI Conf.*, pages 1488–1493, 1997.
28. S.C. Shapiro and O.H. Ismail. Symbol-anchoring in cassie. In S. Coradeschi and A. Saffiotti, editors, *Anchoring Symbols to Sensor Data in Single and Multiple Robot Systems: Papers from the 2001 AAAI Fall Symposium*, pages 2–8. AAAI, 2001.
29. G. Wasson, D. Kortenkamp, and E. Huber. Integrating active perception with an autonomous robot architecture. *Robotics and Automation Journal*, 1999.

Progressive Planning for Mobile Robots
A Progress Report

Lars Karlsson[1] and Tommaso Schiavinotto[2]

[1] Center for Applied Autonomous Sensor Systems, Örebro University
SE-701 82 Örebro, Sweden
lars.karlsson@tech.oru.se
[2] FG Programmiermethodik, Technische Universität Darmstadt
64283 Darmstadt, Germany
schiavin@inferenzsysteme.informatik.tu-darmstadt.de

Abstract. In this article, we describe a possibilistic/probabilistic conditional planner called PTLplan, and how this planner can be integrated with a behavior-based fuzzy control system called the Thinking Cap in order to execute the generated plans. Being inspired by Bacchus and Kabanza's TLplan, PTLplan is a progressive planner that uses strategic knowledge encoded in a temporal logic to reduce its search space. Actions' effects and sensing can be context dependent and uncertain, and the resulting plans may contain conditional branches. When these plans are executed by the control system, they are transformed into B-plans which essentially are combinations of fuzzy behaviors to be executed in different contexts.

1 Introduction

In this article, we describe a conditional planner called PTLplan [21], intended for use on autonomous agents that operate in uncertain environments. A conditional planner does not presuppose that there is complete information about the state of its environment at planning time. However, more information may be available later due to sensing, and this new information can be used to make choices about how to proceed. Therefore, a conditional planner can generate plans that contain conditional branches; an important ability for mobile robot systems that are only able to perform sensing locally. Another important feature of PTLplan is that it can handle degrees of uncertainty, either in possibilistic or in probabilistic terms. The "P" in PTLplan stands for exactly that. Finally, by using strategic knowledge that helps prune away unpromising plan prefixes, PTLplan is able to generate plans efficiently.

We also present some work in progress on how to actually integrate PTLplan with a robot control system. The particular system chosen for this purpose is the Thinking Cap (TC) [26], a close relative of the Saphira architecture [27]. TC is based on fuzzy logic [29]: the world is interpreted in terms of fuzzy properties or predicates, and control actions are represented as fuzzy sets of control values. In TC, fuzzy control rules of the form `IF` *condition* `THEN` *action* are grouped together to form behaviors that are to achieve specific conditions; and behaviors

M. Beetz et al. (Eds.): Plan-Based Control of Robotic Agents, LNAI 2466, pp. 106–122, 2002.

can then be combined to form B-plans (behavior plans). These B-plans will be the link that connects TC and the plans generated by PTLplan.

The first and major part of the paper — sections 2 to 5 — covers different aspects of PTLplan. It is followed by a brief account of the Thinking Cap (Sec. 6), and the integration between the two systems (Sec. 7) including some experiments (Sec. 8).

2 Introduction to PTLplan

PTLplan is a progressive (forward-chaining) planner; it starts from an initial situation and applies actions to that and subsequent resulting situations, until situations where the goal is satisfied are reached. Progressive planning has the disadvantage that the search space is potentially very large even for small problems. The advantage is that the planner can reason from causes to effects, and always in the context of a completely specified situation. The latter makes it possible to apply a technique that can reduce the search space considerably: the use of strategic knowledge that helps prune away unpromising plan prefixes.

PTLplan (and its predecessor ETLplan [20], which did not include uncertainty) builds on a sequential (non-conditional) planner called TLplan [4,5] ("TL" stands for "temporal logic"). Two of the features that makes TLplan interesting are its fairly expressive first-order representation and its good performance due to its use of strategic knowledge. The latter is indicated by its outperforming most other comparable planners in empirical tests, as has been documented in [5]. PTLplan adds four features to TLplan:

- The representation of actions and situations used in PTLplan permit actions that have sensing effects, that is the agent may observe certain fluents (state variables).
- Based on these observations, the planning algorithm can generate conditional plans.
- Degrees of uncertainty, either in possibilistic or probabilistic terms, can be assigned to effects, observations and situations/states.
- Based on these degrees, measures of how likely a plan is to succeed or fail can be computed. In the absence of plans that are completely certain to succeed, PTLplan is still capable of finding plans that, although they may fail, are likely enough to succeed.

PTLplan is the first progressive planner utilizing strategic knowledge to incorporate the features mentioned above, and as indicated by tests, it does so successfully [21].

3 PTLplan: Representation

PTLplan can represent uncertainty either using possibility theory [1] or probability theory, simply by interpreting the connectives used as below.

[1] In possibility theory [13], one assigns degrees of possibility $Pos(E)$ and necessity $Nec(E)$ in the interval [0,1] to (sub)sets of alternative outcomes $E \subseteq \mathcal{E}$ of an event in

Connective	Possibility	Probability	Comment
$\otimes$	min	$\cdot$	"and"
$\oplus$	max	$+$	"or"

Thus, PTLplan can be used both when one is only interested in the relative likelihood of different facts and outcomes, i.e. possibilities, and when one has precise information about probabilities. In this paper, we focus on planning with possibilities, in particular because of the close relation between possibilistic and fuzzy logic [30]. We also sometimes speak about necessity, which is defined as: $Nec(\varphi) = 1 - Pos(\neg\varphi)$.

In PTLplan, the world is viewed as a collection of fluents, which can take on different values. The fluent-value formula `robot-at(table1)=F` means that the fluent has the value `F` (false). If there is no value, then value `T` (true) is implicitly assumed. A fluent-assignment formula such as `robot-at(table1):=F` denotes that a fluent is caused to have a value, due to some action. Finally, a fluent formula is any combination of fluent-value formulae formed with the standard logic connectives and quantifiers.

An action schema consists of a tuple $\langle a, P, R\rangle$ where a is the action name, P is a precondition (a fluent formula, possibly with some free variables) and R is a set of result descriptions. Each result description $r \in R$ is a tuple $\langle C, p, E, O\rangle$ where C is a context condition (a fluent formula) that determines when (in what states) the result is applicable; p is the possibility or probability of the result; E is a set of fluent assignment formulae which specifies the effects of the result; and O is a set of fluent-value-formulae f=v denoting that f is (correctly or incorrectly) observed to have the value v.

Example 1. The following are two examples of action schemas actually used in the mobile robot scenarios we have tested. First, the robot can check whether a door is open.

act: `check-door(d)`
pre: `robot-at(x)∧door(d)∧part-of(x,d)`
context *p* *effects* *observations*
res: `(open(d)=T, 1.0, {}, {open(d)=T}),`
`(open(d)=F 1.0, {}, {open(d)=F})`

Secondly, the robot can enter a room:

act: `enter(y)`
pre: `robot-at(x)∧connected(x,y)∧`
`∃d [ door(d)∧open(d)∧part-of(x,d)∧part-of(y,d) ]`
context *p* *effects* *observations*
res: `(true, 1.0, {robot-at(x):=F, robot-at(y):=T}, { })`

order to indicate the likelihoods of these outcomes (one possible interpretation of the degrees is as upper (Pos) and lower (Nec) bounds on probabilities). An assignment of possibility and necessity degrees can be uniquely represented by a possibility distribution $p : \mathcal{E} \to [0, 1]$ which assigns a value to each individual outcome. Possibility and necessity are then defined as: $Pos(E) = \sup_{e \in E} p(e)$ and $Nec(E) = 1 - \sup_{e \in \overline{E}} p(e)$.

To keep track of what the world looks like at different points in time, we use a model of knowledge and action which is based on the concepts of a situation and an epistemic situation. In short, a situation s describes one possible state of the world at a given point in time, where time is defined in terms of what actions have been executed. A state $state(s)$ is a mapping from fluents to values. Each situation also has an observation set $obs(s)$, which is a set of fluent-value-pairs $\langle f, v\rangle$. The global possibility/probability of a situation s is denoted $p(s)$. This represents the possibility/probability that the specific choice of actions (plan) will lead to s.

An epistemic situation $\overline{s}$, or e-situation for short, is essentially a set of situations with associated possibilities/probabilities that describes the agent's knowledge at a point in time. In this PTLplan is similar to e.g. C-BURIDAN [12] which employs a probability distribution over states. All situations in an e-situation should have the same observation set. The global possibility/probability $p(\overline{s})$ of an e-situation is $p(\overline{s}) = \bigoplus_{s \in \overline{s}} p(s)$.

Example 2. The following is an example of an e-situation where there is uncertainty on whether door `door123` is open or not.

$obs=\{\,\}$
`1.0: { robot-at(corr-E3)=T, open(door123)=T, ...}`
`0.4: { robot-at(corr-E3)=T, open(door123)=F, ...}`

The result of an operator/action A in a situation is defined as follows: for each result description $\langle C_i, p_i, E_i, O_i\rangle$, if context condition C_i is true in s then there is a situation s' resulting from s where the fluents in E_i are assigned their new values (all fluents not in E_i stay the same) and the observations in O_i are made (if $''f{=}v'' \in O_i$, then $\langle f, v\rangle \in obs(s')$). Finally, $p(s') = p(s) \otimes p_i$. For an epistemic situation $\overline{s}$, the result of an action A is determined by applying A to the different situations $s \in \overline{s}$ as above, and then partitioning the resulting situations into new e-situations according to their observation sets.

Example 3. For instance, applying the action `check-door(door123)` to the e-situation above results in two new e-situations:

$obs=\{\,\langle$`open(door123), T`$\rangle\}$
`1.0: { robot-at(corr-E3)=T, open(door123)=T, ...}`

$obs=\{\,\langle$`open(door123),F`$\rangle\}$
`0.4: { robot-at(corr-E3)=T, open(door123)=F, ...}`

The present version of PTLplan explicitly enumerates the situations constituting an e-situation, which apparently does not scale very well. More compact (but equivalent) representations will be investigated.

3.1 Syntax: Conditional Plans

Plans in PTLplan are conditional. This means that there can be points in the plans where the agent can choose between different ways to continue the plan

depending on some explicit condition. Therefore, in addition to the sequencing plan operator (`;`), we introduce a conditional operator (`cond`). The syntax of a conditional plan (C-plans) is as follows:

plan ::= `success` | `fail` | *action* `;` *plan* |
`cond` *branch*`*`
branch ::= `(`*cond* `:` *plan*`)`
action ::= *action-name*`(`*args*`)`

A condition *cond* is a conjunction of fluent-value formulae. The conditions for a branch should be exclusive and exhaustive relative to the potential e-situations at that point in the plan. `Success` denotes predicted plan success, and `fail` denotes failure.

Regarding the semantics of a conditional plan, the applications of a single action and a sequence are defined in the obvious way. The application of a conditional plan element `cond` $(c_1 : p_1) \ldots (c_n : p_n)$ is defined as an application of the branch p_j whose context condition c_j matches with $obs(\overline{s}_i)$ (there should only be one such branch).

4 PTLplan: Strategic Knowledge

In order to eliminate unpromising plan prefixes and reduce the search space, PTLplan (and TLplan before it) utilizes strategic knowledge. This strategic knowledge is encoded as expressions (search control formulae) in an extension of first-order linear temporal logic (LTL) [14] and is used to determine when a plan prefix should not be explored further. One example could be the condition "never enter a new location and then immediately return to your previous location". If this condition is violated, that is evaluates to `false` in some e-situation, the plan prefix leading there is not explored further and all its potential continuations are cut away from the search tree. A great advantage of this approach is that one can write search control formulae without any detailed knowledge about how the planner itself works; it is sufficient to have a good understanding about the problem domain.

LTL is based on a standard first-order language consisting of predicate symbols, constants and function symbols and the usual connectives and quantifiers. In addition, there are four temporal modalities: $\mathcal{U}$ (until), $\Box$ (always), $\Diamond$ (eventually), and $\bigcirc$ (next). In LTL, these modalities are interpreted over a sequence of situations, starting from the current situation. For the purpose of PTLplan, we can interpret them over a sequence/branch of epistemic situations $B = \langle \overline{s}_1, \overline{s}_2, \ldots \rangle$ and a current epistemic situation $\overline{s}_i$ in that sequence. The expression $\phi_1 \mathcal{U} \phi_2$ means that ϕ_2 holds in the current or some future e-situation, and in all e-situations in between ϕ_1 holds; $\Box\phi$ means that ϕ holds in this and all subsequent e-situations; $\Diamond\phi$ means that ϕ holds in this or some subsequent e-situation; and $\bigcirc\phi$ means that ϕ holds in the next e-situation $\overline{s}_{i+1}$.

We let $\mathcal{G}\varphi$ denote that it is among the agent's goals to achieve the fluent formula φ. Semantically, this modality will be interpreted relative to a set of

Algorithm. *Progress(f,$\overline{s}$)*
Case:

1. $f = \mathcal{K}f_1 :\ f^+ := \begin{cases} \texttt{true} \text{ if } K \leq (1 - \bigoplus_{s \in S^-} p_l(s)) \text{ where } S^- = \{s \in \overline{s} | s \models \neg f_1\}, \\ \texttt{false} \text{ otherwise} \end{cases}$
2. $f = \mathcal{O}(f_1{=}v) :\ f^+ := \begin{cases} \texttt{true} \text{ if } \langle f_1, v \rangle \in obs(\overline{s}), \\ \texttt{false} \text{ otherwise} \end{cases}$
3. $f = \mathcal{G}f_1 :\ f^+ := \begin{cases} \texttt{true} \text{ if } s \models f_1 \text{ for all } s \in G, \\ \texttt{false} \text{ otherwise} \end{cases}$
4. $f = f_1 \wedge f_2 :\ f^+ := Progress(f_1, \overline{s}) \wedge Progress(f_2, \overline{s})$
5. $f = \neg f_1 :\ f^+ := \neg Progress(f_1, \overline{s})$
6. $f = \bigcirc f_1 :\ f^+ := f_1$
7. $f = f_1 \mathcal{U} f_2 :\ f^+ := Progress(f_2, \overline{s}) \vee (Progress(f_1, \overline{s}) \wedge f)$
8. $f = \Diamond f_1 :\ f^+ := Progress(f_1, \overline{s}) \vee f$
9. $f = \Box f_1 :\ f^+ := Progress(f_1, \overline{s}) \wedge f$
10. $f = \forall x[f_1] :\ f^+ := \bigwedge_{c \in U} Progress(f_1(x/c), \overline{s})$
11. $f = \exists x[f_1] :\ f^+ := \bigvee_{c \in U} Progress(f_1(x/c), \overline{s})$

Return f^+

Fig. 1. The PTLplan progression algorithm

goal states G, i.e. the set of states that satisfy the goal. We also introduce two new modal operators: $\mathcal{K}\varphi$ ("knows") means that the necessity/probability that the fluent formula φ is true in the current e-situation exceeds some prespecified threshold [2] K, given the information the agent has; and $\mathcal{O}f{=}v$ denotes that f is observed to have the value v in the current e-situation. We restrict fluent formulae to appear only inside the $\mathcal{K}$, $\mathcal{G}$ and (for fluent-value formulae) $\mathcal{O}$ operators. For the formal semantics, see [21].

Example 4. The following is a formula stating "never enter a new location and then immediately return to your previous location".

$$\Box(\neg\exists x[\mathcal{K}(\texttt{robot-at}(x)) \wedge \\ \exists y[\bigcirc(\mathcal{K}(\texttt{robot-at}(y)) \wedge x \not= y \wedge \bigcirc(\mathcal{K}(\texttt{robot-at}(x))))]])$$

In order to efficiently evaluate control formulas, PTLplan incorporates a progression algorithm (similar to the one of TLplan) that takes as input a formula f and an e-situation and returns a formula f^+ that is "one step ahead", i.e. corresponds to what remains to evaluate of f in subsequent e-situations. The progression algorithm is shown in figure 1. (The goal states G in case 3 are fixed for a given planning problem and need not be passed along.) Note that the algorithm assumes that all quantifiers range over a finite universe U.

[2] This implies that outside the scope of the $\mathcal{K}$ operator, the logic is in fact boolean.

Algorithm. PTLplan($\overline{s}, f, g, A, \sigma$) **returns** $\langle plan, succ, fail \rangle$
1. If $\overline{s} \models g$ then return $\langle \texttt{success}, p(\overline{s}), 0 \rangle$.
2. Let $f^+ := Progress(f, \overline{s})$; if $f^+ = \texttt{false}$ then return $\langle \texttt{fail}, 0, p(\overline{s}) \rangle$.
   (return the same if maximal search depth is reached.)
3. For the actions $a_i \in A$ whose preconditions are satisfied in all $s \in \overline{s}$ do:
   a. Let $S^+ = Apply(a_i, \overline{s})$.
   b. For each $\overline{s}_j^+ \in S^+$, let $\langle P'_j, succ'_j, fail'_j \rangle :=$ PTLplan$(\overline{s}_j^+, f^+, g, A, \sigma)$.
   c. If $(\bigoplus_j fail'_j) > 1 - \sigma$ then let $cont_i = \langle \texttt{fail}, 0, p(\overline{s}) \rangle$.
   d. If $|S^+| = 1$, let $cont_i = \langle a_1 \texttt{ ; } P'_1, succ'_1, fail'_1 \rangle$.
   Otherwise let $cont_i = \langle P_i, succ_i, fail_i \rangle$ where $P_i = a_i \texttt{;(cond(}c'_1 \texttt{:} P'_1\texttt{)} \ldots \texttt{(}c'_n \texttt{:} P'_n\texttt{))}$, $succ_i = (\bigoplus_j succ'_j)$, $fail_i = (\bigoplus_j fail'_j)$, and each $c'_j = \bigwedge\{f \texttt{=} v \mid \langle f, v \rangle \in obs(\overline{s}_j^+)\}$.
4. Return the $cont_i = \langle P_i, succ_i, fail_i \rangle$ with the lowest $fail_i$, provided $fail_i \leq (1 - \sigma)$, or otherwise return $\langle \texttt{fail}, 0, p(\overline{s}) \rangle$.

Fig. 2. The PTLplan planning algorithm

5 PTLplan: The Algorithm

PTLplan is a progressive planner, which means that it starts from an initial e-situation and then tries to sequentially apply actions until an e-situation where the goal is satisfied is reached. The algorithm is shown in figure 2. It takes as input an e-situation $\overline{s}$, a search control formula f, a goal formula g (with a $\mathcal{K}$), a set of actions A and a success threshold σ (the failure threshold is $1 - \sigma$). It returns a triple $\langle plan, succ, fail \rangle$ containing a conditional plan, a degree (possibility/probability) of success, and a degree of failure. It is initially called with the given initial e-situation and a search control formula.

Step 1 checks if the goal is satisfied in $\overline{s}$; if this is the case it returns the possibility/probability $p(\overline{s})$ of $\overline{s}$. Step 2 progresses the search control formula; if it evaluates to `false`, the plan prefix leading to this e-situation is considered unpromising and is not explored further. In step 3, we consider the different applicable actions. For each such action, we obtain a set of new e-situations (a). We then continue to plan from each new e-situation separately [3] (b). For those sub-plans thus obtained, if the combined possibility/probability of failure is above the failure threshold, we simply return a fail plan (c). If there was a single new e-situation, we return a simple sequence starting with the chosen action (d). Otherwise, we add to the chosen action a conditional plan segment where branches are constructed as follows. The conditions are derived from the different observations of the different e-situations, and the sub-plans are those obtained in (b). The success and failure degrees of this new plan are computed by combining those of the individual sub-plans. In step 4, finally, we return the best of those plans (i.e. the one with the least failure degree) found in step 3, or a fail plan if the degree of failure is too high.

Example 5. Assume the planner is called with $\mathcal{K}$(`robot-at(room123)`) as the goal (g), which should be achieved with a necessity of 0.6. Let the initial epistemic

[3] It may not always be necessary to explore each new e-situation. If one sufficiently likely branch fails, it would be futile to explore its siblings.

situation $\overline{s}_0$ be the one in example 2, and the action set A consists of those actions in example 1. There is a conjunctive search control formulae f, with example 4 among the conjuncts. The planner starts by exploring the initial e-situation.

The goal g is tested in $\overline{s}_0$, and is not satisfied. The search control formula f is progressed, yielding f_0 which is not `false`. Next, an action is picked from the applicable actions in A: `check-door(door123)`. Other actions may and will also be picked, but we choose to follow this particular choice in this example. Applying `check-door(door123)` to $\overline{s}_0$ yields the two e-situations $\overline{s}_1$ and $\overline{s}_2$ in example 3.

The planner continues with $\overline{s}_1$, with f_0 as search control formula. The goal g has not been achieved yet, and the search control formula f_0 is progressed to f_1 (not `false`). Next, an action is picked: `enter(room123)`. This action results in the single e-situation s_{11}:

$$\left|\begin{array}{l} obs = \{\ \} \\ \texttt{1.0: } \{\texttt{robot-at(room123)=T, open(door123)=T, ...}\}. \end{array}\right|$$

The planner continues with $\overline{s}_{11}$, with f_1 as the search control formula. The goal g has indeed been achieved: `robot-at(room123)` holds in all situations in $\overline{s}_{11}$. Thus, $\langle \texttt{success}, 1.0, 0\rangle$ is returned.

We now go back to e-situation $\overline{s}_1$ and action `enter(room123)`. The result returned from $\overline{s}_{11}$ implies that $\overline{s}_1$ returns $\langle \texttt{enter(room123);success}, 1.0, 0\rangle$.

Finally, $\overline{s}_0$ and `check-door` receives $\langle \texttt{enter(room123);success}, 1.0, 0\rangle$ from $\overline{s}_1$, and $\langle \texttt{fail}, 0, 0.4\rangle$ from $\overline{s}_2$ (that branch never led to the goal). This is combined into the C-plan

```
check-door(door123);
cond (open(door123):enter(room123);success)
     (¬open(door123):fail)
```

with a success possibility of 1.0 and a fail possibility of 0.4. This yields a success necessity of 0.6, as desired. The planner stops and returns this C-plan. End example.

Notice the simplicity of the uncertainty calculations in PTLplan, not only for possibilities but also for probabilities. When projecting the effects of a plan, one will obtain a tree of situations, with the leaves being either success or fail nodes, each one with an associated probability or possibility. One can predict that one will end up in exactly one of these final situations. To calculate the probability of success (failure), one sums up the probabilities of those final situations where the plan has succeeded (failed). For possibilities, one takes the maximum of the possibilities of the final situations with success or failure, respectively.

PTLplan has been implemented in Allegro CommonLisp. The performance of PTLplan has been tested on a number of scenarios encountered in the literature, and compared to some other cutting-edge conditional probabilistic and possibilistic planners, with favorable results [21] (the most difficult problem was solved in half a second). In all the scenarios tested, the use of search control rules had a major impact on planning efficiency. Another major contributing factor was the use of cycle detection — working progressively with completely defined

e-states makes it straight-forward to detect when one revisits a previously explored e-state. This efficiency makes PTLplan an interesting candidate for use for realistic applications. In the rest of this paper, we focus on how PTLplan can be integrated into a complete robotic architecture.

6 Robot Control: The Thinking Cap

The Thinking Cap (TC) [26] is a robot control system based on fuzzy logic [29,16] and fuzzy control. We will here give a bottom-up tour of the system.

Facts about the world are encoded as fuzzy predicates, that is predicates that can take values in the interval [0, 1]. Each predicate is determined by an actual measurement. For instance, a predicate `near` might be determined by the measured distance to a particular door or other object. Predicates often have vague meanings; there is no sharp line between when they apply and when they do not. For instance, the value of the predicate `near` is 1 when the distance to the door is near 0 m, and decreases as the distance increases. Technically, `near` is interpreted as a fuzzy set over the domain of distance measurements. Predicates and measurements are maintained in a unit called the Local Perception Space, which receives and interprets perceptual data and performs self-localization and anchoring (connecting symbols to percepts) [10,11]. Fuzzy predicates can be combined into complex conditions using fuzzy connectives such as conjunction, disjunction and negation.

Control actions are also represented as fuzzy sets, over domains of control values. For instance, `slow-down(50)` could be represented as a fuzzy set over speed changes, including decreases around 0.50 m/s^2. Control actions are connected to combinations of fuzzy predicates by fuzzy control rules of the form `IF condition THEN action`, such as:

```
IF near and not heading-ok THEN slow-down(50).
```

This rule states that in states[4] where the robot is near the door but heading in the wrong way, decreases in speed around 0.50 m/s^2 are desirable, and other changes are not desirable. In states where the robot is not near the door or the heading is correct, all changes are equally desirable, as far is this rule is concerned. Mathematically, the rule represents a desirability function which is a fuzzy set over pairs of states and control values, specifying how desirable it is to select a certain control value when in a certain state. In other words, a desirability function is the fuzzy version of a control policy. Of course, in order to control the robot in a given state, a single control value must be picked out from those desirable; this process is called de-fuzzification.

On the next level, fuzzy control rules are grouped together to form behaviors. Each behavior should have some specific purpose, for instance to enter a room through a door. In addition to the control rules, behaviors also include anchoring of the objects upon which the behaviors operate. For instance, in the behavior `cross(door123)`, the symbol `door123` must be connected to the sensor data in

[4] The term "state" here refers to the state of the information stored in the LPS.

the LPS which correspond to that door. The following is the schema for the door crossing behavior.

```
behavior: cross (door)
parameters: direction=any, travel-speed=180,
\ \ \ \ crossing-speed=100, obstacle-sensitivity=0.1
rule-set:
IF heading-left and rhs-clear and not crossing THEN turn-right(8)
IF heading-right and lhs-clear and not crossing THEN turn-left(8)
IF crossing and not velocity-ok THEN accelerate(target-value)
IF near and not heading-ok THEN slow-down(50)
```

In a behavior, more than one fuzzy control rule can be, and often are, active at the same time. In that case, the output from the different rules are blended using fuzzy set operators (typically intersection). This corresponds to a fusing of the desirability functions of the different rules in order to provide a combined and smaller (i.e. more constrained) desirability function. The fact that rules — and also entire behaviors — can be blended is one of the major strengths of fuzzy control.

Finally, on the highest level of TC there are behavior plans, or B-plans. A B-plan consists of context-behavior rules of the from `IF context THEN behavior`, for instance:

```
IF near(door234) and facing(door234) and not in(room234)
  THEN cross(door234)
```

The value of the context determines how active the behavior is, which in turn determines to what degree the behavior contributes to the control of the robot in different states. Context-behavior rules can be combined using for instance the chaining operator ";", where $CB_1; CB_2$ means that CB_1 is to achieve the context of CB_2, and thus CB_1 tends to be activated before CB_2. A second operator is conjunction "&", which is used to activate two behaviors at the same time. As a B-plan is a combination of behaviors, also it encodes a desirability function. The TC includes a simple planner for automatic generation of B-plans.

When executing a B-plan, the TC does not explicitly remember which steps have been executed, which step is the current one and which steps remain. Instead, the information about what behavior to execute at what time is hidden in the contexts. In addition, several behaviors may be active simultaneously, to varying degrees. For instance, chaining $(CB_1; CB_2)$ does not correspond to a strict sequence. There are often periods when both CB_1 and CB_2 are active simultaneously, to varying degrees, and disturbances in the execution of the plan may cause CB_1 to be activated again, even though it previously appeared to be completed. This kind of "soft" transitions between behaviors are typical for fuzzy behavior-based control, and contributes to making B-plans more robust to disturbances.

A number of values can be extracted from a B-plan in order to monitor progress. The competence value is the disjunction of all the contexts in the B-plan; if it is 0 (or below some threshold), then there is no behavior that is active,

and the B-plan is no longer effective. The conflict value is computed by looking at the desirability of control values in the current state; if there are no values with high desirability, this is probably due to groups of conflicting actions (e.g `turn-left` and `turn-right`) being active simultaneously. Finally, satisfaction is simply the fuzzy value of the goal clause; if 1 (or above some threshold), the goal has been achieved.

7 Integration

In this section, we consider how a conditional plan (C-plan), which the planner can reason about, can be implemented as B-plans. It can be noted that in the division of labor between PTLplan and the Thinking Cap, the latter assumes a high degree of responsibility; it is not merely executing commands from the planner. The actions that PTLplan delivers to TC are more like subtasks to be achieved in order to solve the overall task; subtasks such as being in a particular location adjacent to the present location. TC, already equipped with a good degree of high-level competence, then implements each subtask as a B-plan, and executes it. PTLplan is also responsible for predicting and making certain strategic decisions and seeing to that the necessary information is available, at the conditional branches. Decisions of exactly what behaviors to execute and at what times are the responsibilities of the TC. Finally, the kind of uncertain information PTLplan deals with is mainly incomplete information of strategic importance, such as whether a certain door currently not observable is open or closed. The TC deals with uncertainty in measurements, due to for instance noise and limited resolution.

7.1 From C-Plans to B-Plans

Single actions in the C-plan are converted to a short B-plan, consisting of one or more context-behavior rules. In addition, there is a goal in the B-plan representing the condition the action is supposed to achieve. One example in our mobile robot domain is the action for entering a room through a door, which was presented above. It is converted to a B-plan consisting of several steps, where the last one represents the goal:

```
IF not near(door234) and not in(room234) THEN reach(door234)
IF near(door234) and not facing(door234) and not in(room234)
THEN face(door234)
IF near(door234) and facing(door234) and not in(room234)
THEN cross(door234)
IF in(room234) THEN still(goal)
```

Action sequences are implemented by first implementing the first action as a B-plan, and when that is completed, the remainder are implemented according to the same principle.

Conditional branches in a C-plan are to their nature distinctly different from condition-behavior rules in a B-plans. A condition-behavior rule does not involve any commitment to any course of action in the future; it only concerns what to do at the moment and in the light of locally available information. The next moment, what behavior(s) in the B-plan to execute is reconsidered without any memory of what was done the moment before. A conditional branch, on the other hand, involves a long-term commitment to a specific course of actions, excluding the alternative branches. Therefore conditional branches are implemented as follows. The B-plan leading up to the condition ends when the condition has been determined (typically by some information-gathering action such as `check-door`). The branch with the condition that is satisfied is chosen (this is determined by inspecting the LPS), and the B-plan corresponding to the first action(s) in that branch is implemented.

Fluents in PTLplan are typically mapped to predicates in TC. This mapping is important when building the initial e-situation, and when checking conditions in the C-plans.

Regarding uncertainty, fuzzy predicates are straight-forward to interpret in a possibilistic setting. However, in most cases the uncertainty in TC predicates such as `near` are not relevant to PTLplan. Likewise, uncertainty on the PTLplan level such as whether a door is open or not, is often not due to concept vagueness or even uncertainty in measurements. Rather, is often due to absence of measurements of the entity in questions. Thus, the exchange of uncertainty information tends to be quite limited, at least in the scenarios tried so far. However, in a precursor to the current system, experiments with fuzzy matching were performed which involved possibilistic degrees of matching relevant to the planner [9].

7.2 The Executor

The plan executor/monitor is the process that translates suitable segments of the C-plan to B-plans, implements the B-plans and monitors their progress.

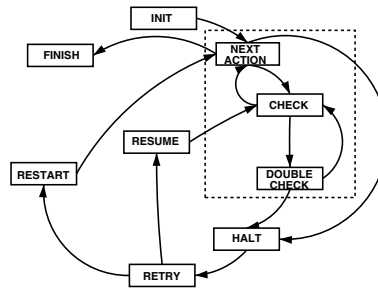


Fig. 3. The states of the executor

The executor can be seen as a finite state automata which is presented in Fig. 3. It starts by receiving a C-plan from the planner (INIT). Then it repeatedly selects the next action to execute (NEXT-ACTION), also selecting branches when appropriate. If the next action is `fail` or `success` the executor finishes (FINISH). Otherwise the action is implemented as a B-plan, with the goal associated with the action as goal for the B-plan. While the B-plan is executing, the goal is checked regularly (CHECK), and when it is achieved, it is time to continue with the next action (NEXT-ACTION again).

While executing, the B-plan may go out of context, in which case the context is double-checked (DOUBLE-CHECK). If the plan then is still out of context, the executor tries to back-track to the last branching in the C-plan and looks for a branch for which the condition is true and the first action is executable. Most of the time, problems can be resolved in this way. However if this does not work, the executor halts the current B-plan (HALT), tries to find an alternative way to implement the current action as a B-plan (RETRY) and then resumes execution (RESUME). If also this fails, the executor informs the planner that it has to try to deliver a new plan (RESTART).

7.3 Discussion: Integration on a High Level

The division of labor between PTLplan and TC permits each module to concentrate on what it does best — PTLplan on solving complicated and long-term tasks involving acquisition of new information, and TC on more short-term planning (which actually could have been done reactively) and plan execution. This division results in a fairly clean interface between the two modules. There are two types of interactions: PTLplan delivers an action to the TC; and TC informs PTLplan that an action has either been completed or has failed. An action is considered failed only when TC has exhausted its possibilities for completing it, and this may involve trying several different B-plans.

In the initial experiments we have made, the relatively simple and clean execution regime presented here has resulted in a fairly robust goal-achieving behavior of our robot. We consider this success to be largely due to the high-level competence of the TC, which allow us to hide many messy details from PTLplan. In the next section, we present the experiments.

8 Experiments

To verify the feasibility of the approach, as well as the usefulness of the planner for practical purposes, a number of experiments were done. The environment was a corridor with a number of offices at the premises of the AASS research center, and the platform used was a Nomad 200 equipped with sonars. All experiments were performed both in the real world and in simulation. Here we describe one of the experiments in detail. The planner was run in possibilistic mode, with the actions `check-door(d)`, `enter(x)` and `move(x)` (the latter is for moving to a new location x without passing though a door.) Only very simple control knowledge was used; the built-in cycle detection was quite sufficient for eliminating most unnecessary movements in the navigation tasks involved.

The experiment had the goal $\mathcal{K}$(`robot-at(E116)` $\vee$ `robot-at(E118)`) where `E116` and `E118` are two offices with door `D116` and `D118`. In the initial state, the robot is at an open area `E115`, which is connected to a corridor `corr-E3` which in turn leads to the two rooms. Both doors may be open (possibility = 1.0), just one of them may be open (possibility = 0.4) or both may be closed (possibility = 0.2). The success threshold was set to 0.8.

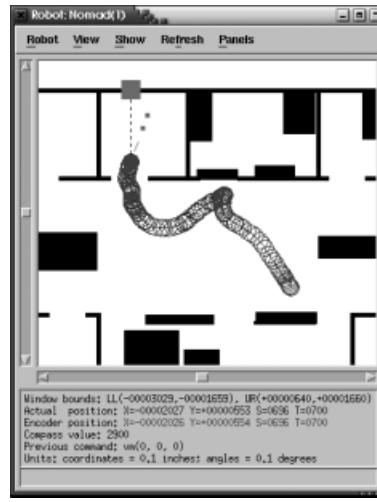


Fig. 4. Second run in simulator, with a trace of the robot's movement. The robots starts in the corridor (lower-right position), tries door D118 (upper-middle) and finishes in room E116 (upper-left)

The experiment was run three times. Each time, PTLplan generated the following C-plan.

```
check-door(D118) ;
 cond ( open(D118)=F : check-door(D116) ;
        cond ( open(D116)=F : fail )
             ( open(D116)=T : enter(E116) ; success ) )
      ( open(D118)=T : enter(E118) ; success )
```

The first time, `D118` was left open, and thus the robot traversed the corridor, checked `D118` and then moved directly into room `E118`.

The second time, `D118` was closed. Here, the robot traversed the corridor, checked `D118`, moved on to `D116`, check it, and entered room `E116` (see Fig 4).

The third time, `D118` was left open, and thus the robot traversed the corridor and checked `D118`. However, after the check but before the robot had entered room `E118`, the door was closed. The robot detected that the plan for entering the room didn't work, and thus back-tracked to the first condition check. It then

selected the other branch, moved on to `D116`, checked that and then entered room `R116`.

Experiments were also performed with a scenario which involved two corridors which may be blocked, and another one which involved checking whether a door is open in order to determine which one of two rooms to enter.

9 Related Work

Planning in partially observable domains has been a hot topic since the mid 90's. There has been a good amount on work on POMDPs (partially observable Markov decision processes), see e.g. [19,8]. Note that this work is on the more general problem of generating policies that maximize utilities, and not plans with sequences and branches that achieve goals.

One of the earliest conditional probabilistic systems originating from classical planning was the partial-order planner C-BURIDAN [12], which had a performance which made in impractical for all but the simplest problems. The partial-order planner MAHINUR [24] improved considerably on C-BURIDAN by handling contingencies selectively. MAHINUR focuses its efforts on those contingencies that are estimated to have the greatest impact on goal achievement. C-MAXPLAN and ZANDER [22] are two conditional planners based on transforming propositional probabilistic planning problems to stochastic satisfiability problems (E-MAJSAT and S-SAT, respectively), and thereby manage to solve the problems comparatively efficiently. On the possibilistic side, there is Guéré's and Alami's conditional planner [17], which also has been demonstrated to have a practical level of performance. Their planner is based on Graph Plan [7]. Relative to these planners, what makes PTLplan interesting is the particular planning technique used — progressive planning with strategic knowledge — and its highly competitive performance [21].

The list of integrated planner-controller systems includes, among others, the robot Shakey [23], which executed plans generated by the classical planner STRIPS [15]; the Aura system [3] with a hierarchical planner; the architecture from LAAS [2] with a temporal planner; the robot Xavier [28] with a partial-order planner on top of a POMDP-based navigation system; the Rhino system [6] using adaptation of reactive plans; and even the Thinking Cap, which includes a simple regressive planner and also has been connected to a BDI system [25]. What characterizes PTLplan-TC in this company is the use of an online conditional planner, and the management of uncertainty that permeates the system.

10 Conclusions

In this paper, we have presented work on the planner PTLplan. It is a progressive planner which utilizes strategic knowledge to reduce its search space. We have described PTLplan's fairly rich representation: actions with context-dependent and uncertain effects and observations, and plans with conditional branches. The semantics of this representation is based on associating different possible

situations with degrees of possibility or probability. We have also described the planning algorithm and how we utilize a temporal logic to encode search control knowledge that can be used to prune unpromising branches in the search tree. PTLplan is based on TLplan [5]. The novel contribution of PTLplan is the introduction of uncertainty into the context of progressive planning with strategic knowledge.

Finally, we have described ongoing work to integrate PTLplan with the fuzzy behavior-based robot control system the Thinking Cap. We have described how C-plans can be implemented and executed as B-plans, and we have also described a series of experiments. Some interesting features of the integration can be recognized. TC is already quite a competent system, and even includes a simple navigational planner. Therefore, instead of specifying exactly what behavior to execute and when, PTLplan delivers a sequence of tasks for TC to solve, and TC then chooses the appropriate behaviors. In addition, there is a similarity between the representation of uncertainty on the two levels: the value of a fuzzy predicate can readily be interpreted in possibilistic terms. Yet, this feature has been of little use in the scenarios presented in this paper, mainly due to the different sources of uncertainty dealt with at the different levels.

As the title of this paper indicates, what is presented here is work in progress. Different scenarios, with a larger repertoire of behaviors, should be tried. The representation of the planner needs to be enriched, for instance with cost estimates or explicit time, and certain aspects of the planner, e.g. the representation of e-situations, need to be made more efficient.

Acknowledgments

This work was funded by the Swedish KK foundation.

References

1. *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, Orlando, Florida, 1999. AAAI Press, Menlo Park, California.
2. R. Alami, R. Chatila, S. Fleury, M. Ghallab, and F. Ingrand. An architecture for autonomy. *The International Journal for Robotics Research*, 17(4):315–337, 1998.
3. Ronald C. Arkin. Integrating behavioral, perceptual, and world knowledge in reactive navigation. *Robotics and Autonomous Systems*, 6:105–122, 1990.
4. F. Bacchus and F. Kabanza. Using temporal logic to control search in a forward chaining planner. In M. Ghallab and A. Milani, editors, *New Directions in Planning*, pages 141–153. IOS Press, Amsterdam, 1996.
5. F. Bacchus and F. Kabanza. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, 116:123–191, 2000.
6. M. Beetz, T. Arbuckle, T. Belker, A.B. Cremers, D. Schulz, M. Bennewitz, W. Burgard, D. Hähnel, D. Fox, and H. Grosskreutz. Integrated, plan-based control of autonomous robots in human environments. *IEEE Intelligent Systems*, 16(5):56–65, 2001.
7. A. Blum and M.L. Furst. Fast planning through planning graph analysis. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, Montreal, 1995. Morgan Kaufmann.

8. C. Boutilier and D. Poole. Computing optimal policies for partially observable decision processes using compact representations. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, Portland, Oregon, 1996. AAAI Press, Menlo Park, California.
9. S. Coradeschi, D. Driankov, L. Karlsson, and A. Saffiotti. Fuzzy anchoring. In *The 10th IEEE Conference on Fuzzy Systems*, Melbourne, 2001.
10. S. Coradeschi and A. Saffiotti. Perceptual anchoring of symbols for action. In IJCAI01 [18].
11. S. Coradeschi and A. Saffiotti. Perceptual anchoring: A key concept for plan execution in embedded systems. In this volume.
12. D. Draper, S. Hanks, and D. Weld. Probabilistic planning with information gathering and contingent execution. In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems*. AAAI Press, Menlo Park, Calif., 1994.
13. D. Dubois and H. Prade. *Possibility theory — An approach to computerized processing of uncertainty*. Plenum Press, 1988.
14. E.A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B*, chapter 16, pages 997–1072. MIT Press, 1990.
15. Richard E. Fikes and Nils J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.
16. S. Gottwald. *Fuzzy sets and fuzzy logic*. Vieweg, Braunschweig, Germany, 1993.
17. E. Guéré and R. Alami. A possibilistic planner that deals with nondeterminism and contingency. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, Stockholm, 1999. Morgan Kaufmann.
18. *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, Seattle, 2001. Morgan Kaufmann.
19. L.P. Kaebling, M.L. Littman, and A.R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1–2):99–134, 1998.
20. L. Karlsson. Conditional progressive planning: A preliminary report. In Brian Mayoh, John Perram, and Henrik Hautop Lund, editors, *Proceedings of the Scandinavian Conference on Artificial Intelligence 2001*. IOS Press, 2001.
21. L. Karlsson. Conditional progressive planning under uncertainty. In IJCAI01 [18].
22. S.M. Majercik and M.L. Littman. Contingent planning under uncertainty via stochastic satisfiability. In AAAI99 [1], pages 549–556.
23. Nils J. Nilsson. Shakey the robot. Technical note 323, SRI Artificial Intelligence Center, Menlo Park, CA, 1984.
24. N. Onder and M.E. Pollack. Conditional, probabilistic planning: A unifying algorithm and effective search control mechanisms. In AAAI99 [1], pages 577–584.
25. S. Parsons, O. Pettersson, A. Saffiotti, and M. Woolridge. Intention reconsideration in theory and practice. pages 378–382.
26. A. Saffiotti. *Autonomous Robot Navigation: A fuzzy logic approach*. PhD thesis, Faculté de Sciences Appliquées, Université Libre de Bruxelles, 1998.
27. A. Saffiotti, K. Konolige, and E.H. Ruspini. A multi-valued logic approach to integrating planning and control. *Artificial Intelligence*, 76(1–2):418–526, 1995.
28. R. Simmons, R. Goodwin, K.Z. Haigh, S. Koenig, J. O'Sullivan, and M. Veloso. Xavier: Experience with a layered robot architecure. *SIGART-Bullentin*, 8(1–4):22–33, 1997.
29. L. A. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.
30. L.A. Zadeh. Fuzzy sets as a basis for a theory of possibilities. *Fuzzy Sets and Systems*, 1:3–28, 1978.

Reasoning about Robot Actions: A Model Checking Approach

Khaled Ben Lamine and Froduald Kabanza

Dept. de Math-Info, Université de Sherbrooke
Sherbrooke, Québec J1K 2R1, Canada
{benlamin,kabanza}@dmi.usherb.ca

Abstract. Mobile robot control remains a difficult challenge in changing and unpredictable environments. Reacting to unanticipated events, interacting and coordinating with other agents, and acquiring information about the world remain difficult problems. These actions should be the direct products of the robot's capabilities to perceive, act, and process information intelligently, taking into account its state, that of the environment, and the goals to be achieved. This paper discusses the use of model-checking to reason about robot actions in this context. The approach proposed is to study behaviors that allow abstract, but informative models, so that a computer program can reason with them efficiently. Model-checking can then be used as a means for verifying and planning robot actions with respect to such behaviors.

1 Introduction

Mobile robot control remains a difficult challenge in unstructured or dynamic environments in which operating conditions are changing and unpredictable. In such instances, the behavior-based robot programming paradigm advocates distributing a robot task among several processes (called behaviors), running concurrently, each dealing with a simpler subtask [1]. A behavior operates by updating one or more robot control parameters at different steps of execution. For instance, a behavior to avoid an obstacle is implemented by a program that changes the robot's heading and speed, at every step of execution, depending on the robot's sensors. A behavior that moves the robot to a target position controls the same parameters, but based on target coordinates instead of the robot's current position. In principle, the latter behavior is not supposed to account for obstacles. Rather it should be the combination of both behaviors that move the robot to the target position while avoiding obstacle. Different behavior-based architectures have been proposed, all agreeing on the concurrent nature of behaviors. They differ in mechanisms used to implement this concurrency and abstraction [2, 3]

Designing behaviors as concurrent processes facilitates their design, but introduces, at the same time, all the fundamental problems of concurrent processes. One typical problem is deadlocks, which appear when the execution of a process is blocked by a condition that was supposed to be enabled by another process.

M. Beetz et al. (Eds.): Plan-Based Control of Robotic Agents, LNAI 2466, pp. 123–139, 2002.

In some cases, deadlocks are normal and intended in the design. For example, an obstacle-avoidance process is blocked while waiting for an obstacle to be signaled by a separate obstacle-detection process. In other cases, deadlocks are not intended at all and they represent design faults that should be fixed. To illustrate, because of programmer oversight, the obstacle-detection process fails to signal a detected obstacle in the obstacle-avoidance process. Another typical problem is livelocks, which appear when the concurrent execution fails to progress towards a desired execution point, for instance, because of pathological cyclic execution by one or more processes. These kinds of design errors are very difficult to detect, yet they can cause fatal robot behaviors.

Similar problems have been being investigated in the area of protocol verification, where successful tools have been developed to automatically detect typical error designs, using model-checking techniques [4]. The basic idea is to simulate protocol rules (or their models) in such a way that the simulation covers all their potential executions, and to check that every simulated execution sequence satisfies a correctness statement. If the statement is specified as a formula in temporal logic, then the problem becomes to check that the execution sequence satisfies the formula; in other words, we check that the execution sequence is a logical model of the formula, that is, model checking. One important problem in this approach is developing efficient modeling and simulation techniques to cover the entire space of execution sequences for given protocol rules. Lesser coverage will yield proportional confidence in the correctness of the protocol rules, but not complete confidence. It may nevertheless be useful as a debugging tool.

We propose adopting similar ideas to improve the design of robot behaviors. As with automatic protocol verification, we would like to check whether the concurrent execution of robot behaviors satisfies a given correctness property. There are three contexts to which we would like to apply this: off-line verification, online verification, and planning. For off-line verification, we would like to simulate a correctness condition to be verified on a set of behaviors, covering the space of their possible executions and checking that they satisfy the correctness condition. For online verification, given a progress condition that should be satisfied by a normal execution of robot behaviors, we would like to track the execution, checking that it does not violate the progress condition. Goals represent tasks for behaviors and may be changed at any time by user requests or by other trigger events. Changes in goals require a reconfiguration of behaviors. Reconfiguration may also be required when the environment context changes, for example, upon failure detection. Behavior planning addresses these reconfiguration problems. Given a goal statement and an environment context, we would like to simulate given behaviors to select combinations or configurations that best suit a given context or goal.

If robot behaviors were like network protocol rules, we could easily solve the above problems through the straightforward adaptation of tools tailored to verify protocol rules. Unfortunately, there are important fundamental differences between protocol rules and robot behaviors. Robot actions (e.g., turn, stop, accelerate, grasp an object or release it) rely on noisy sensors and error-

prone actuators. Although noise also exists in protocols (because of unreliable transmission media), it is more easily abstracted over by checking for corrupted packets at lower levels. Consequently, the problem of verifying protocols deals with higher level transmission rules, such as ensuring logical consistency in the acknowledgment of received packets [4]. Another important difference between robot behaviors and communication protocols is that robot behaviors are more clearly goal-oriented. Hence, the notion of a "goal" should be part of the language used to express correctness statements, so that we can check, for example, if two behaviors involve conflicting goals.

In this paper we discuss some steps relating to the development of more suitable tools for verifying robot behaviors, whether in the context of online verification, off-line verification, or planning. We propose using Linear Temporal Logic (LTL) [5] as the language for specifying properties of robot behaviors. As explained above, depending on the context, such a property will express a design correctness statement, an execution progress condition, or a goal. As these ideas are at the early stage of experimentation, the discussion in this paper remains at a relatively abstract level.

The remainder of the paper is organized as follows. The next section briefly discusses coding robot behaviors using the SAPHIRA control architecture. Section 3 deals with specifying properties of behaviors using LTL. Section 4 discusses a technique for efficiently checking that a behavior trace violates an LTL property and the application of this technique to monitor real-time executions, to detect off-line design errors, and to plan behaviors. We conclude with a discussion on related work.

2 Coding Robot Behaviors

2.1 Robot Platform

We use one indoor Pioneer-1 mobile robot and one outdoor all-terrain Pioneer-1.[1] Figure 1 shows a snapshot of both robots in our lab. Each robot is equipped with 8 sonars (allowing to detect obstacles), a pan-tilt-zoom color camera, an onboard image processing system (allowing recognition of 3 different colors simultaneously), a radio modem (that transmits information from the robot to a remote computer and vice versa), an audio-video transmitter that transmits the camera video output to the computer, and grippers. Information transmitted from the robot to the remote computer via the radio modem mainly consists of sonar readings, robot motor and wheel status, features extracted by the onboard image processor, and the gripper status. Information transmitted from the remote computer to the robot consists of commands on actuators. There are few basic robots control parameters that directly affect the actuators: heading angle (i.e., turning angle for the wheels), speed, camera configuration (pan, tilt, zoom) and gripper configuration (open, closed).

[1] Pioneer-1 mobile robot is a trademark of ActivMedia Inc.

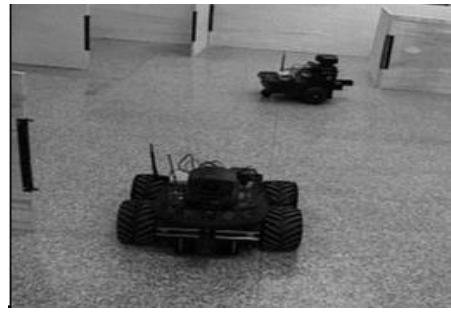


Fig. 1. Pioneer-1 and Pioneer-1 AT robots

2.2 SAPHIRA Behaviors

We program robot behaviors using SAPHIRA architecture [2], which is based on a synchronous model of concurrency. Each behavior is launched with a given priority. Behaviors are executed by a process scheduler, which cycles through them every 100 milliseconds. At every cycle, the scheduler considers each active behavior to determine its output control actions; actions from all processes are combined to determine their joint effect on the robot's actuators, using behavior priorities to resolve conflicts. How control actions are joined cannot be fully described here, but it can be approximated by saying that when behaviors with different priorities affect the same control variable with conflicting values, then the behavior with the highest priority takes precedence. If the behaviors have the same priority, then their conflicting effects on the control variables are merged. For example, if one indicates turning left 45 degrees and the other turning right with 45 degrees, the end result will be moving front (i.e., turn 0 degrees).

To illustrate, let us consider oversimplified specifications of some of the basic robot navigation behaviors that are provided with SAPHIRA (Figure 2). For the sake of concision, we have removed variable declarations that are not crucial for understanding the examples, but that are required for a complete and correct definition. Obstacle-avoidance is implemented by two behaviors: `avoidCollision` (which avoids obstacles close to the robot) and `keepOff` (which veers the robot away from longer obstacles). Both behaviors are similar in description, but react differently as one focuses on close obstacles while the other is detecting long distant obstacles. Behavior `goToPos` moves the robot to a given position. The command `Turn Left get_turn_amount` turns the robot's wheel to the left the number of degrees indicated by `get_turn_amount`. This variable is updated periodically using rules that are declared in the behavior (omitted here for the sake of concision) but roughly the amount is proportional to how far to the left or right the obstacle is. The command `Turn Right` is similar. The command `Speed` sets the robot's speed to a given value in mm/sec.

Variables are local to behaviors and have a fuzzy interpretation. To simplify the examples here, we assume a binary interpretation for variables in the rule antecedents and crisp values for command `Turn` and `Speed`. In `avoidCollision`,

```
 BeginBehavior avoidCollision
  If obstacle_right Then Turn Left  turning_depth
  If obstacle_left  Then Turn Right turning_depth
  If obstacle_front Then Turn preferred_turn turning_depth
 EndBehavior
BeginBehavior keepOff
  If obstacle_right Then Turn Left  turning_depth
  If obstacle_left  Then Turn Right turning_depth
  If obstacle_front Then Turn preferred_turn turning_depth
EndBehavior
BeginBehavior goToPoS
  If gt_too_fast Or gt_too_slow Then Speed gt_speed
  If gt_pos_on_left  Then Turn Left  gt_turn_amount
  If gt_pos_on_right Then Turn Right gt_turn_amount
  If gt_pos_achieved Then Speed 0.0
EndBehavior
```

Fig. 2. Avoid-collision, keep-off and go-to behaviors

`obstacle_left` is set to true when the sensor readings indicate a close obstacle to the left of the robot (say within a range of 2 meters). In `keepoOff`, `obstacle_left` is set to true when the sensor readings indicate there is an obstacle to the left of the robot but further away. An analogue interpretation holds for the other variables. The reader is referred to the SAPHIRA programming manual for a more accurate description of behaviors, but the above abstract examples are sufficient to illustrate our point.

For our examples, the basic control variables affected are robot heading (through command `Turn`) and speed. Consider the situation depicted in Figure 3(a): the robot's goal is to reach the position marked by the large black dot, from a position marked by the large white dot. Presently, there are no obstacles, either close or distant. In this instance, `avoidCollision` and `keepOff` have no effect on the control variables; `goToPos` changes the robot's heading to align it with the target position. In Figure 3(b), the robot has detected a distant obstacle in front, but no close obstacle. In this situation, `keepOff` veers the robot to the right to avoid the obstacle. In Figure 3(c), the robot has detected a distant obstacle in front as well as closer obstacles on the left and right, resulting in conflicting actions. `keepOff` would have the robot veer to the right or left; `avoidCollision` and `goToPos` would have it move front. The SAPHIRA operating system joins the actions, yielding the action of moving straight ahead.

The above behaviors implement obstacle avoidance while moving towards the target position with obstacles. This works quite well in most cases. Yet it is not difficult to find a configuration of obstacles that causes the robot to oscillate between two positions without ever making progress towards the goal (e.g., see Figure 3(d)). In such situations, the robot needs some global path planning. It would, however, run counter to the principle of behavior-based approaches to include a path planner in the obstacle-avoidance behavior itself. Complex

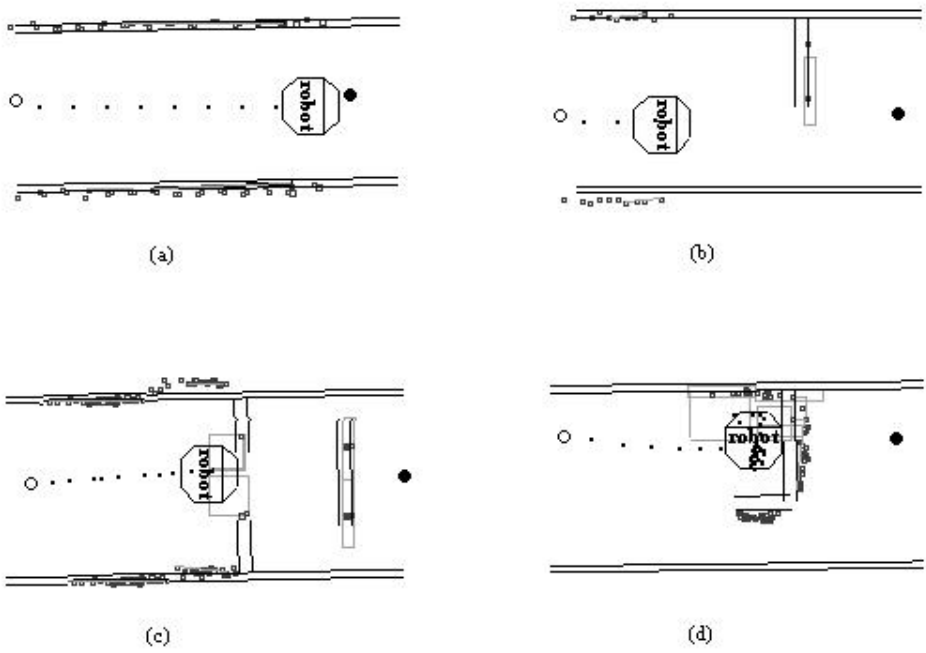


Fig. 3. Robot navigation snapshots

tasks should result from the combination of simple behaviors, not some complex deliberative process.

In keeping with the spirit of behaviors, it would be better to use tools that can monitor the robot's behaviors in order to detect their failures. Such tools can enable us to keep our basic navigation behaviors with the knowledge that they fit some contexts but may fail in some tricky situations. In the latter, a recovery behavior can be invoked to put the robot on the right track. In other words, what is needed is a way to specify progress conditions under which behaviors or combination of behaviors are deemed to be progressing normally. Then, we can have another behavior or process that monitors those progress conditions to send signals when they are violated. That way, with the previous example, we would write a progress condition stating that "as long as the robot has not reached the target position, then its position should progress towards the goal more than 10 m every 60 s." This approach is general and goes beyond navigation behaviors. We would also use it, for example, to control a robot grasping objects by writing progress statement such as "the robot keeps approaching the object only as long as the object is in the visual field of the camera." With higher-level object delivery behaviors, we may say something like "the robot remains within a region until some condition holds, such as until another robot puts an object there required by the first robot."

3 Specifying Properties of Behaviors

A robot state is described by specifying control variables and their respective values. Basic state properties are expressed using predicates, that is, functions over states, that return `true` or `false`, depending on whether or not the property holds. For instance, the predicate `=(x,y)` is defined to return `true` if `x` is equal to `y`. As this example shows, the state argument is implicit in the notation of a predicate. With an appropriate definition, the predicate `in(x,y)` evaluates to `true` in a state in which object `x` or robot `x` is in room `y`. Predicates can have function applications as arguments. For instance, the statement `=(speed(robot),v)` evaluates to `true` in a state in which the robot's speed is `v mm/sec`.

First-order-logic is a predicate language allowing to write more complex logical expressions (called formulas) from predicates by using the logic connectives `&&` (and), `||` (or) , `!` (not), `->` (implies), `forall` and `exists`. Our use of the `forall` quantifier is always bounded by a predicate, that is, we can only quantify on something true of a given a predicate. For instance, we can write

```
forall (x) in(x,Lab3031)
   (<=(position-x,30) &&  >=(position-x,0) &&
    <=(position-y,30) && >=(position-y,0))
```

The predicate `in(x,Lab3031)` here is mandatory. This means for every `x` such that `x` is in `Lab3031`, the position of `x` satisfies the indicated boundary constraints.

First-order logic allows us to write statements that are true about a given state. In order to write statements that are true about behaviors, we need a logic that can express statements about possible execution sequences. LTL is such a language obtained from First-order logic by introducing operators that are applied to formulas to relate them to the future of a given execution sequence or to its past.[2]

3.1 Formal Syntax

The LTL formula formation rules are:

1. a predicate is the most simple LTL formula (including the primitive propositions `true` and `false`);
2. for any LTL formulas `f` and `g`, and time interval `i`, then the following are also LTL formulas:

[2] First-order logic becomes suitable for expressing properties of execution sequences if we include a state argument in predicates. For example, "in(x,Lab3031,s)" would mean that "in(x,Lab3031)" holds in state "s". This extension of First-order logic is known as Situation-Calculus [6]. LTL relates predicates to states in a different way using modal temporal operators, leading to a different approach for evaluating formulas over execution sequences.

(a) `! f` (intuitively: "not `f`"),
(b) `f && g` (intuitively: "`f` and `g`"),
(c) `next i f` (intuitively: "the next state is on the time interval `i` and satisfies `f`");
(d) `f until i g` (intuitively: "on the forwards time interval `i`, `f` holds in every future state up to a state satisfying `g`");
(e) `last i f` (intuitively: "the last state is on the time interval `i` and satisfies `f`");
(f) `f since i g` (intuitively: "`f` holds in every past state up to a state satisfying `g`, on the time interval `i`")
(g) `forall x1 ... xn p f` (intuitively: "for all `x1 ... xn` such that `p` holds, then `f` holds", where `p` is a predicate involving `x1 ... xn` as free variables, and `f` is a formula also possibly involving those free variables);
(h) Parenthesis may be used to resolve ambiguities.

The future (or forwards) operators (`next` and `until`) refer to future of an execution sequence, that is, the subsequence rooted from the current state. The past (or backwards) operators (`last` and `since`) are like their mirror reflections referring to the history of an execution sequence, that is, the subsequence starting from initial state for the execution sequence and ending in the current state. The arguments of a temporal operator are a time interval and one or two formulas. The time interval is noted `[i,j]`, where `i` is the starting time, assuming the time in the current state is 0, and `j` is the ending time. The ending time is noted `?` when it is infinite. This interpretation holds going forwards if the interval is associated to a future operator, or going backwards if it is associated to a past operator. In our case, the time unit is generally set to the length of a cycle for the SAPHIRA operating system (i.e., 100 milliseconds.) The following abbreviations are standard:

- `f || g` is equivalent to `!(!f && !g)` (intuitively: "`f` or `g`").
- `f->g` is equivalent to `!f || g` (intuitively: "`f` implies `g`")
- `eventually i f` is equivalent to `true until g` (intuitively: "`f` eventually holds in some future state on the time interval `i`").
- `always i f` is equivalent to `! eventually i !f` (intuitively: "`f` holds in every future state on the time interval `i`").
- `previously i f` is equivalent to `true since i f` (intuitively: "`f` holds on some past state on the time interval `i`").
- `alwaysPreviously i f` is equivalent to `! previously i !f` (intuitively: "`f` holds in past future state on the time interval `i`").
- `exists x1 ... xn p f` is equivalent to `! forall x1 ... xn p !f` (intuitively: "there exists `x1, ..., xn` such that if `p` holds then `f` holds");

3.2 Examples of LTL Formulas

1. The LTL formula `always (0,?) !(active(B1) && active(B2))` states that behavior `B1` and behavior `B2` must never be active at the same time.

2. The formula

```
always [0,?] (! in(robot,Lab3031) ->
              next [0,?] (in(robot,Lab3031) ->
                          eventually [0,2]
                           always [0,100] active(B1)))
```

   means that, once the robot enters Lab3031, then behavior `B1` must be active within 2 time units and remain active during 100 time units.
3. The formula

```
always [0,?] (forall (x) in(x,Lab3031)
               (grasping(robot,x) -> requested(x)))
```

   means that, in Lab3031, the robot should only grasp requested object.
4. The formula

```
always [0,?]
 (((last [0,?] ! in(robot,Lab3031)) &&
   in(robot,Lab3031)) ->
  eventually [0,2] (always [0,100] active(B1)))
```

   is just another way of expressing the same statement as in (2) above.
5. The formula

```
always [0,?] ((alwaysPreviously [0,10)] stalled()) ->
              next [0,?] active(B3))
```

   means that, if the robot is stalled since 10 time units, then behavior `B3` must be active in the next state.
6. The formula

```
always [0,?]((last [0,?] clost() &&
              last [0,?] last [0,?] clost() &&
              last [0,?] last [0,?] last [0,?] clost()) ->
             next [0,?] active(B4))
```

   means that, if we have three consecutive losses of communication with the robot (`clost`), then behavior `B4` must be active in the next state.

3.3 Formal Semantics

Reactive behaviors, such as obstacle-avoidance, are cyclic by nature, with no predefined terminating point. Such a cyclic execution unwinds into an infinite execution sequence. Thus the semantics of LTL formulas is defined by considering infinite execution sequences. In this case, a terminating execution sequence is represented by an equivalent one obtained by replicating the terminal state infinitely.

The LTL interpretation rules are recursively defined in Figure 4. This function takes three arguments, respectively, an LTL formula `h`, an execution sequence `E`, and a state `s` on `E`. It returns `true` if `h` holds in `s`; otherwise, it returns `false`. This must be only regarded as a specification of the semantics rule, not as an effective procedure for checking LTL formulas over an execution sequence. Since we assume an infinite execution sequence, the "algorithm" does not actually terminate. In fact, we realized that it facilitates understanding when the semantics rules are given that way in an algorithmic style. Below, we explain simple modifications to this specification to obtain an effective method for checking LTL formulas.

The basic case is with predicates. The generic function `holdPredicate` takes a predicate and state as arguments and then calls a domain-dependent predicate-evaluation function that returns `true` if the predicates holds in the state, `false` otherwise.

The recursive case is domain-independent and implements the interpretation of logical connectives and temporal operators. A formula `! f` holds on `s` if `f` does not hold on `s`. A conjunctive formula holds on `s` if each conjunct holds on `s`.

A formula of the form `next i f` holds on `s` if `f` holds on the successor of `s`, and this successor appears in the interval `i`; note that every state has a successor since the sequence is infinite. Function `succ(s,E)` returns the state immediately following `s` on the execution sequence `E`; `dur(s,s1,E)` returns the duration between state `s` and state `s1` on the execution sequence `E`; `lb(i)` returns the lower bound of a time interval `i`; `ub(i)` returns the upper bound.

The interpretation rule for `until` means `f until i g` is satisfied in the current state if the interval `i` is active (i.e., its lower bound is 0) and `g` is satisfied in the current state, otherwise if `f` is satisfied in the current state and `(f until j g)` is satisfied in the next state, with `j` representing a reduction of `i` with the time elapsed between the current and the next state. For a situation in which the upper-bound of an interval is `?`, `?-d` reduces to `?` for any duration `d`.

The interpretation rules for `previously` and `since` are analogue to, respectively, `next` and `until`, but it goes backwards, using the function `pred(s,E)` to access to the state before `s` on the execution sequence `E` (this yields `null` if `s` is the initial state).

Finally, `forall x1 ... xn p f` holds in a state if `f` holds in that state for every instantiation of `x1, ..., xn` that makes `p` true in the state.

4 Monitoring and Planning Behaviors

4.1 Monitoring Robot Behaviors

The SAPHIRA operating system cycles through behaviors every 100 ms to determine their effects on robot control parameters, resulting in a state change every 100 ms. However, abstract properties do not require such a fine granular level of state sampling. For instance, if we are checking whether or not a robot enters a given room, we may check this at a periodicity in terms of minutes rather than

```
hold(h,E,s) {
 if (h is a predicate) return holdPredicate(h,s);
 if (h is of the form (! f)) return (! hold(f,E,s));
 if (h is of the form (f && g)) return (hold(f,E,s) && hold(g,E,s));
 if h is of the form (next i f)
    return ((lb(i) <= dur(s,succ(s,E),E)) &&
            (ub(i) >= dur(s,succ(s,E),E)) &&
            hold(f,succ(E,s),E));
 if (is of the form (f until i g)) {
    let s1 = succ(E,s);
    let j  = [max(0,lb(i) - dur(s,s1,E)), max(0,ub(i) - dur(s,s1,E))];
    if (j == [0,0]) return hold(g,E,s);
    else return ((lb(i)==0 && ub(i) != 0 && hold(g,E,s)) ||
                 (hold(f,E,s) && hold((f until j g),E,s1)));}
 if (h is of the form (last i f))
      return ((pred(s) == void) ||
              ((lb(i) <= dur(pred(s,E),s,E)) &&
               (ub(i) >= dur(pred(s,E),s,E)) &&
               hold(f,E,pred(s))));
 if (is of the form (f since i g)) {
    let s1 = pred(E,s);
    if (s != null)
       let j = [max(0,lb(i) - dur(s1,s,E)), max(0,ub(i) - dur(s1,s,E))];
    else let j = [0,0];
    if (j == [0,0]) return hold(g,E,s);
    else return ((lb(i)==0 && ub(i) != 0 && hold(g,E,s)) ||
                 (hold(f,E,s) && hold((f since j g),E,s1)));}
 if (is of the form (forall x1 ... xn p f)) {
    let result = true;
    for every p1 obtained from p by instantiating (x1, ..., xn)
    such that holdPredicate(p1,s) {
      let f1 obtained from f by applying the same instantiation;
      result = result && hold(f1,E,S);}
    return result;}}
```

Fig. 4. LTL Semantics

milliseconds. For this, we define *state-sampling frequency* parameter, at which states are sampled. This depends on the type of progress conditions being monitored. Not all robot control variables have to be tracked. Only those relevant to atomic propositions in LTL progress conditions need to be involved. For this, we introduce a function *state-reflector* that tracks relevant state features.[3]

[3] The SAPHIRA operating system maintains a basic robot's control state in a structure accessible to user-written programs. This includes the current robot position, speed, heading, and sonar readings. The status of behaviors (active or suspended) is also accessible to user-written programs and hence can be traced.

By keeping a trace of the robot state during an execution, we check whether or not the robot's execution up to the current point of execution does not violate given LTL progress conditions (for example those in Section 3.2). However, rather than explicitly applying the LTL interpretations, we use a more efficient approach by checking the LTL progress conditions *on the fly*, keeping information relevant to the history in past formulas rather than explicitly in trace, and delaying the evaluation of future formulas in the next state. This technique is known as "progressing LTL formulas over a sequence of states". It was first introduced for future operators, in the TLPLAN planning system [7]. Here we extend it to formulas with past operators.

4.2 Progressing Formulas

The idea is to keep track of the following: (1) the *current state*, (2) a trace of execution to the current state and (3) a set of *delayed LTL formulas*, each corresponding to a "delayed" progress condition in the current state. Each of these is updated at every execution cycle (i.e., at every state change).

Intuitively, a delayed formula is one which would have been evaluated in the next successor state by a recursive call to `hold`, but instead, had its evaluation postponed. The delayed formula is also called "progressed formula", because it is progressed from one state to another.

Initially, the current state is the initial state of the robot (as given by the *state reflector*); the trace is empty; the set of delayed formulas is the given set of progress conditions.

At a current state, this is how each of the three components are updated. The new state, successor of the current one, is automatically given by the *state reflector*. The new trace is obtained from the current one by appending the current state.[4] For each LTL progress condition, a corresponding new LTL formula is obtained by invoking a "delayed" evaluation of the LTL-interpretation procedure on the current formula, current trace and current state, assuming the new state as successor and a time duration between them equal to the *state-sampling period*.

By "delayed" evaluation of LTL-interpretation procedure, we mean an invocation of the procedure in Figure 4, such that each recursive call to `hold` involving the new state as argument just returns its input formula without further evaluation; that is, the evaluation is delayed by just returning the formula. All other recursive calls are made, leading either to `true` or `false`. With simplifications, the final result is either `true`, `false` or a formula whose main connective is `next`. This is the new, updated LTL progression condition.

That way, the formula `f'` returned by the delayed evaluation of `f` expresses would have to be satisfied by any execution sequence starting from the new state

[4] This is a naive approach for storing traces. We are investigating more efficient approaches, where we can exploit syntactic information about the LTL progress conditions being monitored to only store partial, but sufficient information about the trace.

in order for a corresponding execution just one step earlier in the current state to satisfy `f`. A proof of this claim simply follows from the fact that this is a delayed evaluation of the LTL semantics rules.

Hence, if the delayed formula is `true` this means that any future execution is satisfactory; we can stop monitoring the corresponding progress condition since it becomes valid from this point of execution. On the other hand, if the delayed formula is `false`, this means that we have a violation of the progress condition; we must thus send a notification to a handling behavior.

If the delayed formula is other than `true` or `false`, this means that so far the execution is progressing normally. There may still be conditions requiring eventual achievement in the future, but none of them has been violated so far.

It is interesting to note that a formula like `(eventually (0, ?) p)`, for any proposition `p`, is never violated by an execution trace. As long as the current trace does not contain a state satisfying `p`, this will be progressed to `(eventually (0, ?) p)`. Intuitively, as long as we have not encountered `p`, we still have a chance of meeting such a state later. It is only at the end of the execution sequence that we can conclude that `p` is not satisfied. A trace that does not contain a state satisfying `p` does not violate `(eventually (0, ?) p)`, but it does not satisfy it either.

In contrast, if we have the formula `(eventually (0, 10) p)`, then at every step, this will be progressed into a similar formula, but with the upper bound of the time interval decremented by the duration between the current and next state. Soon or later, we will reach a state in which 10 time units have elapsed from the initial state, yet without having a met a state satisfying `p` (if we met a state satisfying `p` the delayed formula would have been `true`). This will be progressed to `false`, thus indicating a failure of the progress condition.

Properties that are only violated by infinite executions (or cyclic executions in practice) are usually called liveness properties [5]. They are conveyed by `until` or `eventually` operators with an unbound time-intervals. The opposite are safety properties, which are conveyed by `until` or `eventually` operators with bounded-time intervals, and `always` operators regardless of their time-intervals. In fact, an `until` operator with an unbounded-time actually conveys both a safety property (because its first formula component must be maintained `true`) and a liveness property (because its second component must be eventually made `true`, without a deadline).

4.3 Planning Robot Behaviors

Monitoring is about checking that runtime traces do not violate progress conditions. Planning deals with checking that predicted, simulated future behaviors would continue satisfying those progress conditions. That way, the robot can anticipate failures to re-configure its behaviors.

While a runtime execution is deterministic, a simulation involves nondeterministic guesses. This is so because the effects and preconditions of actions in future states depend, in part, on real-time conditions unknown in the current

state.[5] Hence, when simulating behaviors, we must account for all possibilities; which yields a simulation graph rather than a single execution sequence.

In order to simulate behaviors efficiently, we need a *state transition model* that hides details of behaviors that are irrelevant to progress conditions being monitored. The state-transition model specifies preconditions for the execution of a robot's actions and their effects. Specifying it is one of the most challenging hurdles in our approach. Currently, it is specified independently using STRIPS or ADL action descriptions [7], but we are investigating approaches that would make it possible to ground the state-transition model in the source code of SAPHIRA behaviors.

The simulation is essentially a search process through the space of possible future executions of robot behaviors at a level of abstraction described by the state-transition model. In this context, it is helpful to classify progress conditions into goals (i.e., tasks that the robot must achieve) and search control strategies that guide the search process [7]. LTL search-control formulas can be understood as additional constraints on the behaviors reflecting behaviors that are irrelevant to some goals and, hence, that should not be simulated.

Liveness properties pose an additional difficulty when planning for cyclic behaviors. A property like `(eventually (0, ?) p)` expresses a goal of eventually achieving `p`, without a deadline. Formula-progression alone cannot check the violation of such properties, unless some simplifying assumptions are made, such as fixing one goal state [7], or approximating goals without deadlines with goals with arbitrary deadlines [8]. [6] Cyclic behaviors can be dealt with by explicitly checking that goals of achievement without deadline are not violated by cyclic behaviors, but this introduces an additional complexity [10].

4.4 Summary

The key software components composing our approach are summarized in Figure 5. The *state extractor* implements an interface between SAPHIRA and LTL, by extracting state features that are relevant to predicates appearing in the LTL specification of progress conditions. The *Runtime monitor* checks given progress conditions over runtime behavior. This is done incrementally, by labeling each state of the trace with the corresponding LTL formulas, using LTL formula progression. A formula progressed to `false` in a given state means that the corresponding condition is violated by the trace. For each violated formula, a signal

[5] Since SAPHIRA uses a synchronous model of concurrency, no nondeterminism is entailed in the interleaving of processes as in the asynchronous case. With UNIX processes, for example, we would have to guess how the UNIX operating system would interleave them.

[6] This is the approach taken by Drummond and Bresina, although using a goal language that is a subset of LTL and a model-checking procedure less general than formula-progression [8]. In general, assuming that only propositions are negated, using goals with deadlines limit formulas to only safety properties and hence completely verifiable by using formula-progression alone [9].

is generated indicating which formula is violated and at what point of the trace. The signal is then handled by an ad-hoc behavior.

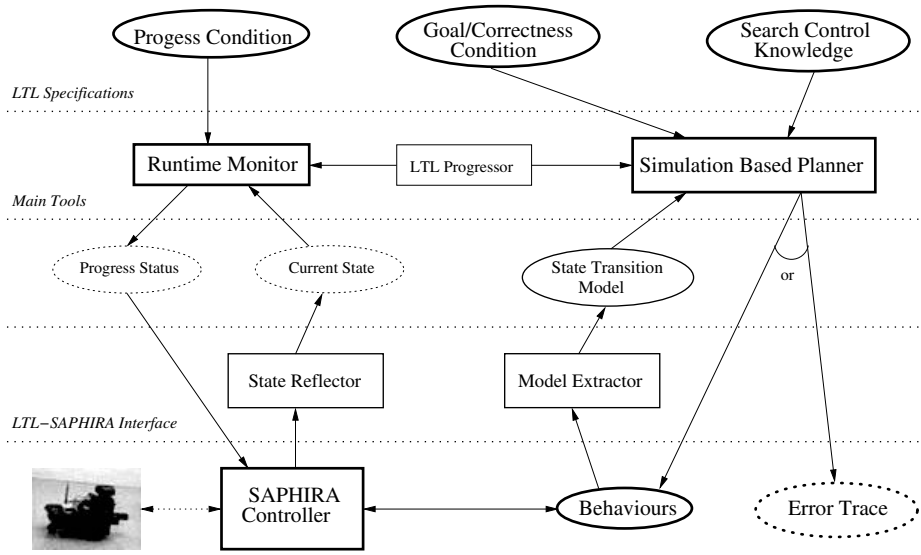


Fig. 5. Software Architecture

On the other hand, from a state-transition model, we simulate execution sequences from a given context; a simulation-based planner evaluates LTL goal statements over those sequences, allowing to determine combinations of behaviors whose future execution would be expected to best satisfy the goal, that is, planned behaviors. In a different mode, the planner accepts correctness criterion to validate over simulated behaviors.

As the implementation currently stands, the monitor and the planner are operational, but both still need validation on more realistic applications. This is where most of the current implementation effort is being driven. Besides the automatic extraction of state-transition models from SAPHIRA behaviors that is mentioned above, we are also working on a more efficient approach for storing past information during a simulation. This is essential when planning is done with a goal that is a mixture of past and future operators, because a trace has to be stored somehow in every state. As most states share traces, the question is to represent this efficiently into the planner. This is less problematic in monitoring because there is always one single trace.

5 Conclusion

Pure behavior-based approaches are simply reactive. They involve no explicit representation of the robot's goals, plans, or internal "world model." Goals are only implicit in the situation-action coupling and plans emerge as one action is executed in a way that it triggers or deactivates another. This is done in a modular way, in which simple behaviors are run concurrently to achieve complex behaviors. This simplicity facilitates the design of immediate real-time responses to sensed events. This is well illustrated by the remarkable performance of behavior-based approaches in robot navigation, using very simply coded behaviors. On the other hand, deliberative approaches use explicit representation of the world model constructed from sensory data and the actions executed by the robot emerge from the interplay of an explicit planner (or other formal reasoning component) on the world model and given robot goals. One clear advantage of this approach is that behaviors can be planned automatically. Having both features in a robot control architecture leads to hybrid control architectures.

Our motivation is not to propose yet another hybrid architecture. Instead, we are proposing tools to both monitor and plan robot behaviors using the same basic technique, that is, checking that execution sequences generated from runtime or simulated behaviors satisfy LTL correctness statements. Experimenting with this approach in the SAPHIRA architecture facilitates its implementation because SAPHIRA is already a deliberative architecture allowing symbolic representation of the robot world model. In particular, all basic feature of the robot's control state are available symbolically, such as robot speed, heading, and activation status of processes. Hence, it is easy to define LTL propositions on top of these features and other user-defined control variables.

Our approach can also be integrated with other behavior-based architectures, provided it is possible to extract symbolic state information. Schönherr et al. describe a method doing that for connected behavior-based architectures [11]. More precisely, their approach makes it possible to extract symbolic facts characterizing the activation of behaviors. These facts could be considered as proposition from which LTL progress conditions could be defined, which enable the use of our formal monitoring tool. In a similar vein, but with respect to planning, Nicolescu and Mataric discuss an idea about relating behaviors (coded in Ayllu architecture) to STRIPS-like operators [12]. This is compatible with our approach since it makes it possible to simulate Ayllu behaviors through the application of STRIPS operators to obtain state sequences for LTL goal checking.

In the earlier stage of this work, we started with a fuzzy version of LTL [13]. This was motivated by the use of fuzzy control in SAPHIRA. In the end, we introduced past operators and planning, and at the same time abstracted over fuzzy-control, so as to keep uniformity in the language used for planning and monitoring. Haslum is also exploring techniques similar to ours for monitoring and predicting control systems for unmanned aerial vehicles [14]. Our approach can also be related, to a limited extent, to a research program being conducted by Alur *et al.* [15]. They are experimenting the use of automata-theoretic methods

to synthesize robot behaviors that are constructively proven to satisfy some logical properties.

Acknowledgment

This work is supported by the Canadian Natural Sciences and Engineering Research Council (NSERC). We also would like to thank the reviewers for helpful comments.

References

1. Arkin, C.: Behavior-Based Robotics. The MIT Press (1998)
2. Konolige, K., Myers, K., Ruspini, E., and Saffiotti, A.: The Saphira architecture: A design for autonomy. Journal of Experimental and Theoretical Artificial Intelligence **9** (1997) 215–235
3. Werger, B.: Ayllu: Distributed port-arbitrated behavior-based control. In: Proc. of 5th International Symposium on Distributed Autonomous Robotic Systems (DARS). (2000) 24–35
4. Holzmann, G.: Design and Validation of Computer Protocols. Prentice-Hall (1991)
5. Manna, Z., and Pnueli, A.: The Temporal Logic of Reactive and Concurrent Systems. Springer-Verlag (1991)
6. De Giacomo, G., Lespérance, Y., and Levesques, H.: Congolog, a concurrent programming language based on the situation calculus. Artificial Intelligence **121** (2000) 109–169
7. Bacchus, F., and Kabanza, F.: Using temporal logics to express search control knowledge for planning. Artificial Intelligence **116** (2000) 123–191
8. Drummond, M., and Bresina, J.: Anytime synthetic projection: Maximizing probability of goal satisfaction. In: Proc. of 8th National Conference on Artificial Intelligence (AAAI 90), MIT Press (1990) 138–144
9. Barbeau, M., Kabanza, F., and St-Denis, R.: Synthesizing plant controllers using real-time goals. In: Proc. of 14th International Joint Conference on Artificial Intelligence (IJCAI), Morgan Kaufmann (1995) 791–798
10. Kabanza, F., Barbeau, M., and St-Denis, R.: Planning control rules for reactive agents. Artificial Intelligence **95** (1997) 67–113
11. Schönherr, F., Cistelecan, M., Hertzberg, J., and Christaller, T.: Extracting situation facts from activation value histories in behavior-based robots. In: Proc. of Joint German/Austrian Conference on AI, LNAI Vol. 2174. (2001) 305–319
12. Nicolescu, M., and Mataric, M.: Extending behaviour-based systems capabilities using an abstract behaviour representation. In: Proc. of AAAI Fall Symposium on Parallel Cognition. (2000) 27–34
13. Ben Lamine, K., and Kabanza, F.: History checking of temporal fuzzy logic formulas for monitoring behavior-based mobile robots. In: Proc. of the 12th IEEE International Conference on Tools with Artificial Intelligence. (2000) 312–319
14. Haslum, P.: Models for prediction. In: In IJCAI Workshop on Planning under Uncertainty and Incomplete Information. (2001) 8–17
15. Alur, R., *et al.*: A framework and architecture for multirobot coordination. In: Proc. 7th International Symposium on Experimental Robotics. (2001) 289–300

Lifelong Planning for Mobile Robots

Maxim Likhachev[1,2] and Sven Koenig[1]

[1] College of Computing, Georgia Institute of Technology
Atlanta, GA 30332-0280, USA
[2] School of Computer Science, Carnegie Mellon University
Pittsburgh, PA 15213, USA
{mlikhach,skoenig}@cc.gatech.edu

Abstract. Mobile robots often have to replan as their knowledge of the world changes. Lifelong planning is a paradigm that allows them to replan much faster than with complete searches from scratch, yet finds optimal solutions. To demonstrate this paradigm, we apply it to Greedy Mapping, a simple sensor-based planning method that always moves the robot from its current cell to a closest cell with unknown blockage status, until the terrain is mapped. Greedy Mapping has a small mapping time, makes only action recommendations and can thus coexist with other components of a robot architecture that also make action recommendations, and is able to take advantage of prior knowledge of parts of the terrain (if available). We demonstrate how a robot can use our lifelong-planning version of A* to repeatedly determine a shortest path from its current cell to a closest cell with unknown blockage status. Our experimental results demonstrate the advantage of lifelong planning for Greedy Mapping over other search methods. Similar results had so far been established only for goal-directed navigation in unknown terrain.

1 Introduction

Intelligent systems have to adapt their plans continuously to changes in the world or their models of the world. Examples include internet agents and mobile robots. Many systems replan from scratch. However, this can be very inefficient in large domains with frequent changes and thus keeps the systems idle, which is often unacceptable. Fortunately, the changes are usually small. For example, an internet agent might learn that a network node went down or a robot might sense a few previously unknown obstacles. This suggests that a complete re-computation of the best plan is wasteful since some of the previous planning results can be reused. Replanning and plan-reuse methods replan faster than planning from scratch because they reuse information from previous plan-construction processes to avoid the parts of the new plan-construction process that are identical to the previous one. Examples include case-based planning, planning by analogy, plan adaptation, transformational planning, planning by solution replay, repair-based planning, and learning search-control knowledge. These methods have been used as part of systems such as CHEF [18], GORDIUS [43], LS-ADJUST-PLAN [16], MRL [21], NoLimit [48], PLEXUS [4], PRIAR [20],

M. Beetz et al. (Eds.): Plan-Based Control of Robotic Agents, LNAI 2466, pp. 140–156, 2002.

and SPA [19]. There are also more specialized systems that replan paths for mobile robots [17]. We recently introduced lifelong-planning methods, that differ from many existing replanning and plan-reuse methods methods in that they are guaranteed to find optimal plans, and applied them to symbolic planning [22,30]. The term lifelong planning was chosen in analogy to lifelong learning [46].

In this paper, we use lifelong planning on mobile robots because they often have to replan as their knowledge of the world changes. Researchers from mobile robotics have, so far, exploited lifelong planning in unknown terrain only for goal-directed navigation, where (Focussed) D* [45] convincingly demonstrates its advantages since it solves its search tasks with a speedup of one to two orders of magnitude(!) over repeated A* searches, which is important to avoid the robots being idle. However, D* is very complex and thus hard to understand, analyze, and extend. For example, while D* has been widely used as a black-box method, it has not been extended by other researchers. We believe that two achievements are required to make lifelong-planning techniques more popular in mobile robotics. First, one needs to devise simpler lifelong-planning techniques. In previous work, we have therefore developed D* Lite [25]. It is simpler and consequently easier to understand and analyze than D*, yet is at least as efficient. Second, one needs to demonstrate the advantages of lifelong-planning methods, such as D* Lite, for additional navigation tasks. In this paper, we therefore demonstrate how D* Lite can be applied to mapping, which is the first demonstration of how lifelong-planning methods can be used in the context of mapping. By demonstrating the versatility and computational benefits of lifelong planning for mobile robots, we hope that this underexploited technique will be used more often in mobile robotics.

2 Our Lifelong-Planning Methods

Although lifelong-planning methods are not widely known in artificial intelligence and control, some researchers have developed lifelong-planning versions of uninformed search methods in the algorithms literature under the name "incremental search." An overview can be found in [15]. We, on the other hand, have developed lifelong-planning versions of A*, thus combining ideas from the algorithms literature and the artificial intelligence literature.

We have developed Lifelong Planning A* [24] for the task of repeatedly finding a shortest path from a given start vertex to a given goal vertex while the edge cost of a graph change. Lifelong Planning A* combines ideas from DynamicSWSF-FP [38] and A* [36]. The first search of Lifelong Planning A* (and D* Lite without heuristic search) is the same as that of A* but all subsequent searches are much faster. Lifelong Planning A* produces at least the search tree that A* builds. However, it achieves a substantial speedup over A* because it reuses those parts of the previous search tree that are identical to the new search tree, and uses a clever way of identifying these parts. The simplicity

of Lifelong Planning A* allows us to prove various properties about it and show a strong similarity to A*.

We have developed D* Lite [25] for the task of moving a robot from a given start vertex to a given goal vertex while the edge costs of a graph change. It always moves the robot on a shortest path from its current vertex to the goal vertex, and replans the shortest path when the edge costs change. D* Lite combines ideas from Lifelong Planning A* and (Focussed) D* [45]. It implements the same navigation strategy as D* but is simpler and consequently easier to understand and analyze, yet is at least as efficient.

The pseudo code of D* Lite is shown in Figure 1. It uses the following notation. S denotes the finite set of vertices of the graph. $s_{start} \in S$ denotes the current vertex of the robot, initially the start vertex, and $s_{goal} \in S$ denotes the goal vertex. $Succ(s) \subseteq S$ denotes the set of successors of $s \in S$. Similarly, $Pred(s) \subseteq S$ denotes the set of predecessors of $s \in S$. $0 < c(s, s') \leq \infty$ denotes the cost of moving from s to $s' \in Succ(s)$. The heuristics $h(s, s')$ estimate the distance between vertex s and s'. D* Lite requires that the heuristics satisfy $h(s, s') \leq c(s, s')$ for all vertices $s \in S, s' \in Succ(s)$ and $h(s, s'') \leq h(s, s') + h(s', s'')$ for all vertices $s, s', s'' \in S$. The heuristics are guaranteed to satisfy these properties if they have been derived by relaxing the graph, which will almost always be the case.

The procedure ComputeShortestPath() of D* Lite finds a shortest path from the current vertex of the robot to the goal vertex. It does this by calculating g-values that represent the distance of a vertex to the goal vertex. The robot can then follow a shortest path by always moving to an adjacent vertex so that it greedily decreases the g-value of its current vertex. (In the pseudo code, we have included a comment on line {33} how the robot can detect that there is no path but do not prescribe what it should do in this case.) ComputeShortestPath() performs an incremental A*-like search from the goal vertex toward the current vertex of the robot to calculate the g-values, using the heuristics to guide its search. We can prove a variety of properties of ComputeShortestPath(), including its correctness, efficiency, and similarity to A* [23]. For example, we say that ComputeShortestPath() expands a vertex when it calculates its g-value by executing lines {16-28}, which is similar to A* expanding a vertex. ComputeShortestPath() expands every vertex at most twice before it returns but usually expands many fewer vertices. It is efficient because it performs lifelong searches and calculates only those g-values that have been affected by cost changes or have not been calculated yet in previous searches. It is also efficient because it performs heuristic searches and calculates only the g-values of those vertices that are important to determine a shortest path from the current vertex of the robot to the goal vertex.

In the following, we first describe a particular mapping paradigm that we call Greedy Mapping. Then we explain how to model it as a graph-search problem and use D* Lite to solve it.

The pseudocode uses the following functions to manage the priority queue: U.Top() returns a vertex with the smallest priority of all vertices in priority queue U. U.TopKey() returns the smallest priority of all vertices in priority queue U. (If U is empty, then U.TopKey() returns $[\infty; \infty]$.) U.Insert(s, k) inserts vertex s into priority queue U with priority k. U.Update(s, k) changes the priority of vertex s in priority queue U to k. (It does nothing if the current priority of vertex s already equals k.) Finally, U.Remove(s) removes vertex s from priority queue U.

procedure CalculateKey(s)
{01} return $[\min(g(s), rhs(s)) + h(s_{start}, s) + k_m; \min(g(s), rhs(s))]$;

procedure Initialize()
{02} $U = \emptyset$;
{03} $k_m = 0$;
{04} for all $s \in S$ $rhs(s) = g(s) = \infty$;
{05} $rhs(s_{goal}) = 0$;
{06} U.Insert$(s_{goal}, [h(s_{start}, s_{goal}); 0])$;

procedure UpdateVertex(u)
{07} if $(g(u) \neq rhs(u)$ AND $u \in U)$ U.Update$(u, CalculateKey(u))$;
{08} else if $(g(u) \neq rhs(u)$ AND $u \notin U)$ U.Insert$(u, CalculateKey(u))$;
{09} else if $(g(u) = rhs(u)$ AND $u \in U)$ U.Remove(u);

procedure ComputeShortestPath()
{10} while (U.TopKey()<CalculateKey(s_{start}) OR $rhs(s_{start}) > g(s_{start}))$
{11} u = U.Top();
{12} k_{old} = U.TopKey();
{13} k_{new} = CalculateKey(u);
{14} if($k_{old} < k_{new}$)
{15} U.Update(u, k_{new});
{16} else if $(g(u) > rhs(u))$
{17} $g(u) = rhs(u)$;
{18} U.Remove(u);
{19} for all $s \in Pred(u)$
{20} if $(s \neq s_{goal})$ $rhs(s) = \min(rhs(s), c(s, u) + g(u))$;
{21} UpdateVertex(s);
{22} else
{23} $g_{old} = g(u)$;
{24} $g(u) = \infty$;
{25} for all $s \in Pred(u) \cup \{u\}$
{26} if $(rhs(s) = c(s, u) + g_{old}$ OR $s = u)$
{27} if $(s \neq s_{goal})$ $rhs(s) = \min_{s' \in Succ(s)}(c(s, s') + g(s'))$;
{28} UpdateVertex(s);

procedure Main()
{29} $s_{last} = s_{start}$;
{30} Initialize();
{31} ComputeShortestPath();
{32} while $(s_{start} \neq s_{goal})$
{33} /* if $(rhs(s_{start}) = \infty)$ then there is no known path */
{34} $s_{start} = \arg\min_{s' \in Succ(s_{start})}(c(s_{start}, s') + g(s'))$;
{35} Move to s_{start};
{36} Scan graph for changed edge costs;
{37} if any edge costs changed
{38} $k_m = k_m + h(s_{last}, s_{start})$;
{39} $s_{last} = s_{start}$;
{40} for all directed edges (u, v) with changed edge costs
{41} $c_{old} = c(u, v)$;
{42} Update the edge cost $c(u, v)$;
{43} if $(c_{old} > c(u, v))$
{44} if $(u \neq s_{goal})$ $rhs(u) = \min(rhs(u), c(u, v) + g(v))$;
{45} else if $(rhs(u) = c_{old} + g(v))$
{46} if $(u \neq s_{goal})$ $rhs(u) = \min_{s' \in Succ(u)}(c(u, s') + g(s'))$;
{47} UpdateVertex(u);
{48} ComputeShortestPath();

Fig. 1. D* Lite (optimized version)

3 Greedy Mapping

Mapping is an important task for mobile robots and a large number of mapping methods have been developed for them, both in robotics and in theoretical computer science [10,29,14,27,3,9,13,31,33,39,2,11,12,28,40,37,7,34,35]. A good overview is given in [41]. In this paper, we study Greedy Mapping, a simple sensor-based planning method that discretizes terrain into cells and then always moves the robot from its current cell to a closest cell with unknown blockage status, until the terrain is mapped. It is greedy because its plans quickly gain information but do not take the long-term consequences of the movements into account. Greedy Mapping is probably one of the first mapping methods that come to mind when empirical robotics researchers quickly need to implement a mapping method, and versions of it have been used on robots by different research groups [47,26,42]. It has the following desirable properties:

- **Theoretical Foundation:** Greedy Mapping has a solid theoretical foundation that allows one to characterize its behavior analytically. For example, it is guaranteed to map terrain under realistic assumptions and its plan-execution time can be analyzed formally. In fact, one can prove that its mapping time is reasonably small [26].
- **Simple Integration into Robot Architectures:** Greedy Mapping is simple to implement and integrates well into complete robot architectures. For example, it is robust with respect to the inevitable inaccuracies and malfunctions of other architecture components and does not need to have control of the robot at all times. This is important because search methods should provide robots only with advice on how to act and work robustly even if that advice is ignored from time to time [1]. For example, if a robot has to recharge its batteries during mapping, then it might have to preempt mapping and move to a known power outlet. Once restarted, the robot should be able to resume mapping from the power outlet, instead of having to return to the cell where mapping was stopped (which could be far away) and resume its operation from there. Greedy Mapping exhibits this behavior automatically. Similarly, Greedy Mapping does not expect that the robot faithfully follows its advice. Obstacle avoidance, for example, can change the proposed movement to avoid that the robot gets too close to obstacles. In this case, the robot ends up at a location different from where Greedy Mapping expected it to move. This is not a problem for Greedy Mapping since it automatically resumes its operation from the new location.
- **Prior Knowledge:** Greedy Mapping takes advantage of prior knowledge about parts of the terrain (if available) since it uses all of its knowledge about the terrain when it determines which cell with unknown blockage status is closest to the robot and how to get there quickly. It does not matter whether this knowledge was previously acquired by the robot or provided to it.
- **Distributed Search:** Mapping tasks can be solved with several robots that each run Greedy Mapping and share their maps, thereby decreasing the mapping time. Cooperative mapping is a currently very active research area [8,44].

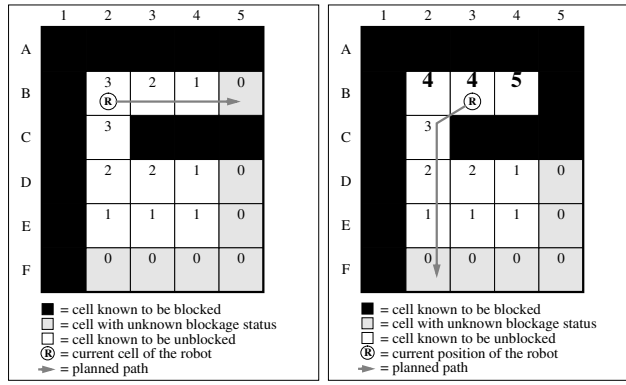


Fig. 2. Example Mapping Task

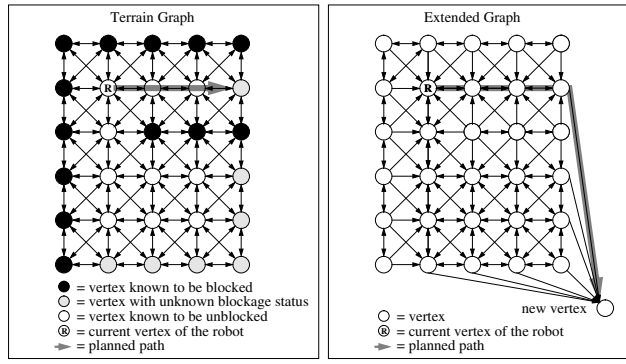


Fig. 3. Graphs for the Example Mapping Task

Greedy Mapping frequently needs to determine a shortest path from the current cell of the robot to a closest cell with unknown blockage status. In the following, we show how D* Lite can be used to implement Greedy Mapping efficiently. This is the first demonstration of how lifelong-planning methods can be used to map unknown terrain, resulting in an efficient implementation of Greedy Mapping.

4 Representation as Graph-Search Problem

The mapping problem can be formulated as a graph coverage problem, for example, by imposing a regular eight-connected grid over the terrain. The vertices of the directed graph correspond to cells and are either blocked or unblocked. The robot moves along the edges of the graph but cannot move onto blocked vertices. The robot knows the graph but initially does not know exactly which vertices are blocked. It can utilize initial knowledge about the blockage status of

vertices in case it is available. For example, Figure 2 (left) shows a terrain with some prior knowledge about the blockage status of vertices, and Figure 3 (left) shows the corresponding grid. Black cells (vertices) are known to be blocked and white cells are known to be unblocked. The blockage status of grey cells is currently unknown and needs to be determined by the robot. The robot has an on-board sensor that reports the blockage status of vertices close to it, including the blockage status of the vertices adjacent to its current vertex. Greedy Mapping remembers this information and uses it to always move the robot on a shortest unblocked path from its current vertex to a closest vertex with unknown blockage status. In Figure 2 (left), the cells are labeled with their distance to a closest cell with unknown blockage status. The robot follows a shortest unblocked path from its current vertex to a closest vertex with unknown blockage status by always moving to an adjacent vertex so that it greedily decreases the distance of its current cell. The arrow shows such a path. While following this path, the robot is guaranteed to discover the blockage status of at least one vertex with unknown blockage status and is thus guaranteed to make progress with mapping. This is so because it will eventually observe the blockage status of the vertex it navigates to if it does not gain information about the blockage status of other vertices earlier. Figure 2 (right) shows that a robot with a sensor range of two cells observes that cell B5 is blocked after it moves one cell to the east along the planned path. Whenever the robot gains information about the blockage status of vertices, the shortest unblocked path from its current vertex to a closest vertex with unknown blockage status can change, either because the closest vertex with unknown blockage status changes or newly discovered blockages change the path to it. Similarly, whenever the robot deviates from the planned path, the shortest unblocked path from its current vertex to a closest vertex with unknown blockage status changes because its current vertex is no longer on the planned path. In both cases, Greedy Mapping needs to recalculate a shortest unblocked path from the current vertex of the robot to a closest vertex with unknown blockage status. Figure 2 (right) shows the new path.

5 Solving the Graph-Search Problem with D* Lite

Greedy Mapping could be implemented with any graph-search method. In the following, we explain the advantages of implementing it with D* Lite. We have already explained that D* Lite uses two different ways of speeding up its searches. It is a heuristic search method and thus uses heuristic knowledge to focus its search and speed it up. It is also a lifelong-planning method and thus uses information from previous searches to speed up its current search. In the following, we explain why both heuristic and lifelong searches have the potential to recalculate a shortest unblocked path from the current vertex of the robot to a closest vertex with unknown blockage status faster than other search methods.

– **Heuristic Search:** D* Lite can use heuristic information to speed up its search, in form of estimates of the distances between two vertices of the

graph. The heuristics could estimate the distance from a vertex to a closest vertex with unknown blockage status. However, this distance is hard to estimate since there are often several vertices with unknown blockage status in different directions around the robot. Instead, the heuristics of D* Lite estimate the distance from the current vertex of the robot to a given vertex. D* Lite can use these heuristics because it effectively performs a backward search from all vertices with unknown blockage status to the current vertex of the robot. These heuristics can easily be obtained since vertices correspond to cells. We calculate an estimate of the distance between two vertices on an eight-connected grid as the maximum of the absolute differences of their x and y coordinates of the corresponding cells. This results in consistent heuristics that are for eight-connected grids what Manhattan distances are for four-connected grids. For example, the heuristic value of vertex B4 is two in Figure 2 (left) because the estimated distance from the current vertex of the robot (B2) to vertex B4 is calculated as the maximum of the absolute difference of their x coordinates, which is two, and the absolute difference of their y coordinates, which is zero. On the other hand, the heuristic value of vertex B4 is only one in Figure 2 (right) since the estimated distance from the new vertex of the robot (B3) to vertex B4 is only one.

– **Lifelong Search:** D* Lite can also use information from previous searches to speed up its search. Figure 2 demonstrates how reusing information can potentially save search effort in the context of Greedy Mapping. The left part of the figure shows the distances of all cells to a closest cell with unknown blockage status. The right part of the figure shows the same distances after the robot moved one cell to the east along the planned path and gained information about the blockage status of cell B5. All but three distances (shown in bold) remain unchanged and therefore do not need to be recomputed even though the path changed completely. This suggests that reusing information from previous searches can potentially reduce the search time of heuristic search methods for Greedy Mapping. For example, when replanning the shortest path in Figure 2 (right), D* Lite only expands the three vertices whose distances to a closest cell with unknown blockage status have changed (namely B2, B3, and B4). On the other hand, A* expands five vertices even with the best possible tie-breaking criterion and thus is less efficient than D* Lite.

We now explain how D* Lite can be used to implement Greedy Mapping. We introduce a new vertex (that becomes the goal vertex of D* Lite) and then construct the so-called extended graph as follows: First, the extended graph contains all edges from the graph that corresponds to the terrain (in the following called the terrain graph), except for those edges that go from vertices that are known to be unblocked or potentially unblocked to vertices that are known to be blocked. This ensures that the planned path is unblocked. (The extended graph contains edges of the terrain graph that go from vertices that are known to be blocked to other vertices. This is important in case the robot has, due to sensor or position uncertainty, mistakenly classified an unblocked vertex as blocked and

then, due to actuator uncertainty, deviates from the planned path and reaches this vertex. The robot believes that it can leave this vertex only if the edges of the extended graph that go from it to other vertices have not been deleted.) Some of the edges just described are deleted from the extended graph (by setting their cost to infinity) when the robot discovers additional blocked cells. Second, the extended graph also contains edges that go from any vertex with unknown blockage status that can be reached with one edge traversal from vertices with known blockage status to the new vertex. This ensures that the planned path reaches a vertex with unknown blockage status and, from there, the new vertex. All of these edges have cost one. Some of the edges just described are added to or deleted from the extended graph (by setting their cost to one or infinity, respectively) when the robot discovers the blockage status of additional cells. Figure 3 shows the extended graph (right) that corresponds to the terrain graph (left) that models the terrain in Figure 2 (left). A shortest path on the extended graph from the current vertex of the robot to the new vertex corresponds to a shortest unblocked path on the terrain graph from the current vertex of the robot to a vertex with unknown blockage status, and vice versa. Thus, Greedy Mapping can determine a shortest unblocked path in the terrain graph from the current vertex of the robot to a closest vertex with unknown blockage status by using D* Lite to find a shortest path in the extended graph from the current vertex of the robot to the new vertex. This way it finds the path in the extended graph that is shown in Figure 3 (right). This path corresponds to the path in the terrain graph that is shown in Figure 3 (left) and the path in the terrain that is shown in Figure 2 (left).

6 Integration into Robot Architectures

We integrated Greedy Mapping into a multi-task autonomous robot architecture called MissionLab [32], which is a version of the Autonomous Robot Architecture (AuRA) [6]. AuRA is a hybrid system that consists of a schema-based reactive system at the low level and a deliberative system based on finite state automata at the high level. The reactive component consists of primitive behaviors called motor schemata [5] that are grouped into behavioral assemblages. Each behavior of a behavioral assemblage produces its own recommendation for how the robot should move, in form of a vector. The robot then moves in the direction of the weighted average of all vectors. We utilize that Greedy Mapping makes only action recommendations and thus can coexist with other components of a robot architecture that also make action recommendations. This allows us to implement map building as a behavioral assemblage of three behaviors that takes as parameters the bounds of the area that the robot needs to map: GreedyMapping is a behavior that directs the robot towards a closest cell with unknown blockage status; AvoidObstacles is a behavior that repels the robot from obstacles; and Wander is a behavior that injects some noise. The weight of AvoidObstacle and the distance within which obstacles affect the robot are set depending on the size of the grid. For grids with small (that is, high-resolution) cells, the weight can be

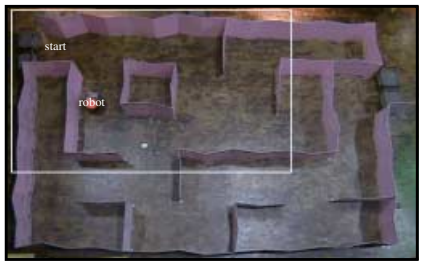


Fig. 4. Maze

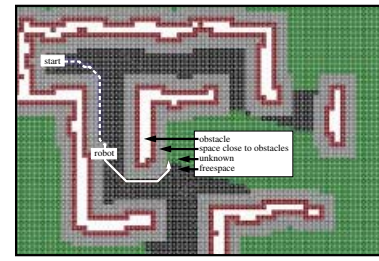


Fig. 5. Screen Shot of Learned Map

set to zero since Greedy Mapping can directly take the obstacles into account. For grids with large cells, the weight can be set to a positive value while the distance within which obstacles affect the robot is made small in comparison to the grid size. This ensures that the robot successfully navigates around small obstacles. The weight of Wander is configured similarly.

We used Greedy Mapping both on a Nomad 150 with a Sick LMS200 laser scanner and in simulation in conjunction with a simple 8-connected grid, the cells of which had a size of 10 centimeters by 10 centimeters. All processing was performed on-board the robot on a Toshiba Pentium MMX 233 MHz laptop running Redhat 6.2 Linux. The robot interleaved sensing, planning, and movement. Sensing consisted of a full 180 degree scan with the laser scanner. Initially, all cells of the grid were marked as having an unknown blockage status. The cells that corresponded to detected obstacles were marked as blocked. Obstacles were surrounded by unblocked cells with a large cost, to bias the robot away from them. The other cells swept by the sensor were marked as unblocked. Cells at distance one from obstacles had traversal cost ten, cells at distances two or three from obstacles had traversal cost five, and cells at larger distances from obstacles had traversal cost one. Planning found a shortest path from the current cell of the robot to a closest cell with unknown blockage status, the first action of which was executed. Then, the cycle repeated until the blockage status of all cells had been observed or the shortest path from the current cell of the robot to a closest cell with unknown blockage status had infinite cost.

We used the robot to map a maze of size 28 by 20 feet that we constructed out of polystyrene insulation on the ground floor of our building. We let the robot map the maze five times. All five experiments were successful. Figure 4 shows a top view of the maze. Figure 5 shows a snapshot of the map during map building, together with the shortest path from the current cell of the robot to a closest cell with unknown blockage status. The part of the maze that corresponds to the part of the map shown in the screen shot is outlined in white in Figure 4.

The version of Greedy Mapping used in this paper assumes that there is neither position nor sensor uncertainty. The assumption that there is no position uncertainty makes Greedy Mapping well suited for outdoor navigation in conjunction with GPS. This assumption was not justified in our experiments since

we used a simple dead-reckoning technique to estimate the location of the robot. The map shown in Figure 5 shows some of the resulting inaccuracies. However, the runs were not long enough for this to become a problem. The assumption that there is no sensor uncertainty was justified. The Sick laser scanner is highly accurate and has sufficient resolution and range accuracy.

7 Experimental Evaluation

In the previous section, we demonstrated that Greedy Mapping is easy to implement and integrate into complete robot architectures. We now demonstrate the advantage of D* Lite over other search methods for implementing Greedy Mapping. We thus compare D* Lite against D*. We also compare it against D* Lite without heuristic search, D* Lite without lifelong search (that is, an A* search toward the current cell of the robot), and D* Lite without lifelong and heuristic search (that is, a breadth-first search toward the current cell of the robot). Since all search methods move the robot in the same way, we only need to compare their total planning time. Since the actual planning times are implementation-dependent, we instead use three performance measures that all correspond to common operations performed by D* Lite and thus heavily influence its planning time: the total number of vertex expansions (which is larger than the total number of expanded vertices if some vertices are expanded more than once), the total number of heap percolates (exchanges of a parent and child in the heap), and the total number of vertex accesses (for example, to read or change their values).

- We first study to which degree D* Lite outperforms D*. We perform experiments in 25 randomly generated terrains of size 64 by 25 that are represented as eight-connected grids and resemble office environments. The robot has no prior knowledge of the terrain. We use the MissionLab simulation to be able to average the results over several runs. We varied the sensor range of the robot to simulate both short-range and long-range sensors. For example, if the sensor range is four, then the robot can sense all blocked cells that are up to four cells in any direction away from the robot as long as they are not blocked from view by other blocked cells. Figure 6 shows the three performance measures for D* as percent difference relative to D* Lite. (Thus, D* Lite always scores zero.) The figure reports not only the means of the three performance measures but also the corresponding 95 percent confidence intervals to demonstrate that our conclusions are statistically significant. D* Lite performs better than D* with respect to all three measures, justifying our claim that it is at least as efficient as D*. (In fact, our experiments show that it is even a bit more efficient than D*.)
- We now study to which degree the combination of lifelong and heuristic searches that D* Lite implements outperforms lifelong or heuristic searches individually. Figure 7 shows an example for sensor range one, where the robot has some prior knowledge of the terrain, both before and after the robot has

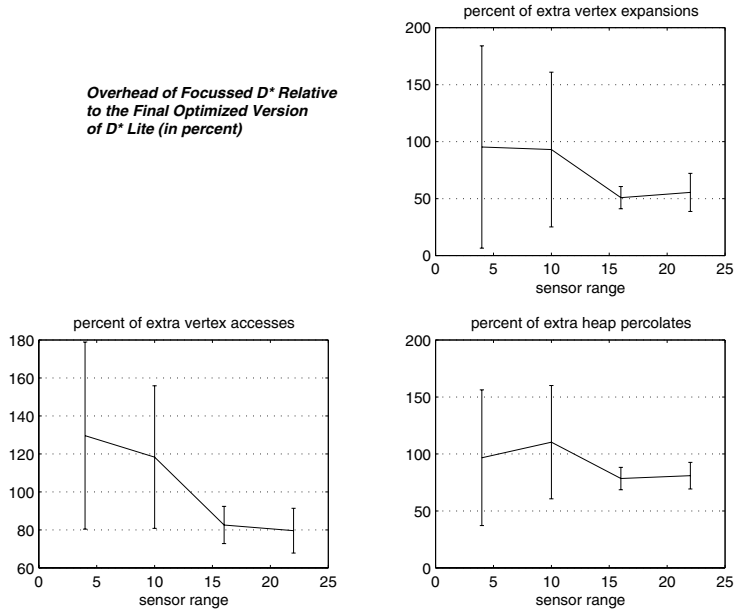


Fig. 6. Comparison of D* Lite and D*

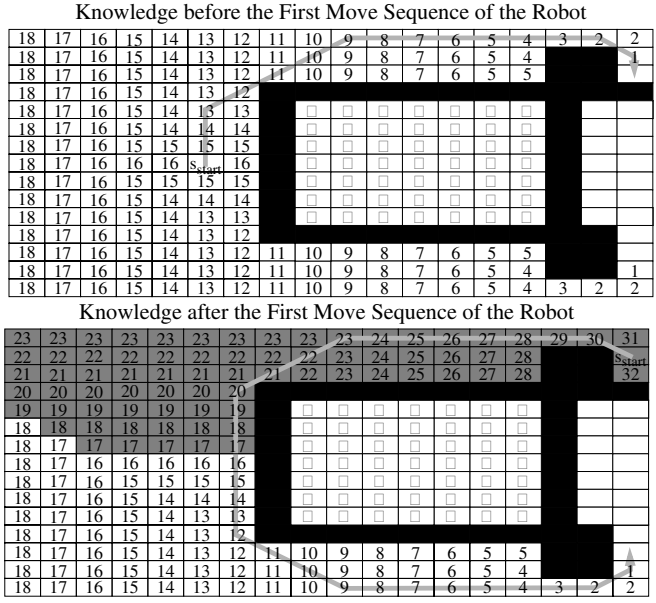


Fig. 7. Illustration of the Behavior of Different Versions of D* Lite (1)

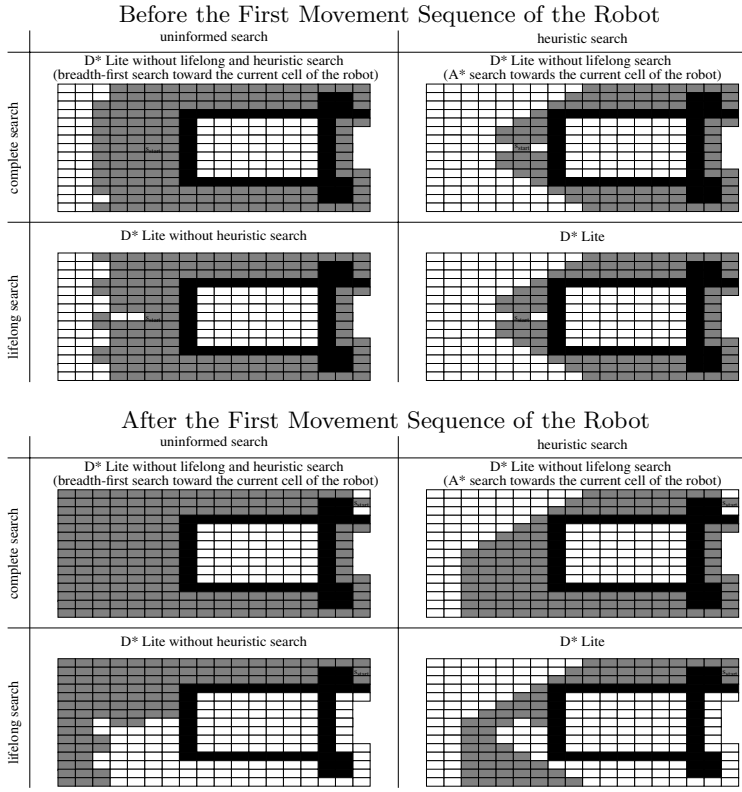


Fig. 8. Illustration of the Behavior of Different Versions of D* Lite (2)

moved along the path and discovered the blockage status of an additional cell. Cells are empty if their blockage status is unknown. The arrows show a shortest path from the current cell of the robot to a closest cell with unknown blockage status. Cells are labeled with their distance to a closest cell with unknown blockage status. Cells whose distances have changed are shaded gray. Notice that not all cells have changed their distance and that some of the changed distances are irrelevant for recalculating the path. Figure 8 shows the cells in gray that are expanded by the different versions of D* Lite. The first search of D* Lite (or D* Lite without heuristic search) expands exactly the same cells as D* Lite without lifelong search (or D* Lite without lifelong and heuristic search) if the search algorithms break ties between vertices with the same f-values suitably. The small differences are due only to the different tie-breaking behavior. On the second search (that is, when previous search results are available), a heuristic search outperforms an uninformed one, and a lifelong search outperforms a complete one. The figures also illustrate that the combination of heuristic and lifelong search performed by D*

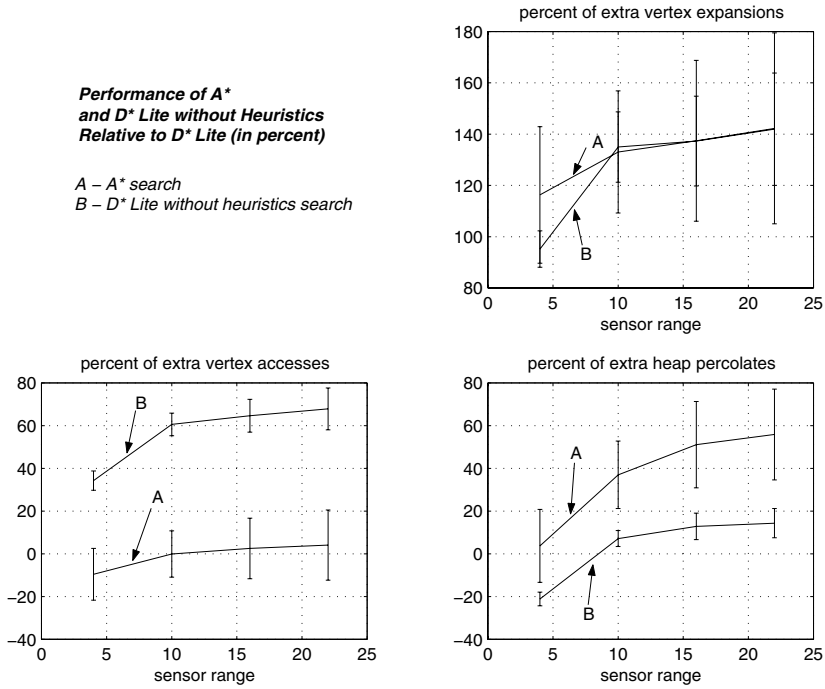


Fig. 9. Comparison of Different Versions of D* Lite

Lite decreases the number of expanded cells even more than either a heuristic or lifelong search individually. To test whether these results are statistically significant, we perform again experiments in 25 randomly generated terrains of size 64 by 25 that are represented as eight-connected grids and resemble office environments. The robot has no prior knowledge of the terrain. Figure 9 shows the three performance measures for D* Lite without heuristic search and D* Lite without lifelong search as percent difference relative to D* Lite. We decided not to include D* Lite without lifelong and heuristic search because it performs so poorly that graphing its performance becomes a problem. As can be seen, the number of vertex expansions of D* Lite is always much smaller than that of the other two algorithms. This also holds for the number of heap percolates and vertex accesses, with the exception of sensor range four for the heap percolates. The advantage of D* Lite over the two other search methods seems to increase as the sensor range increases, that is, the larger the number of cells is whose blockage status the robot can sense without moving. This implies that the advantage of D* Lite increases if the robot uses sensors with longer ranges or discretizes the terrain in a more fine-grained way. This is important since laser scanners tend to be the sensors of choice for mapping and one often prefers fine-grained terrain resolutions. Only for the number of vertex accesses is the difference between D* Lite and

D* Lite without lifelong search statistically not significant although, for the larger sensor ranges, the mean of the number of vertex accesses is smaller for D* Lite than for D* Lite without lifelong search.

8 Conclusions

In this paper, we have explained how lifelong planning applies to mapping of unknown terrain, a new application of lifelong planning. Our results show that the combination of lifelong and heuristic search that D* Lite implements speeds up the planning time of Greedy Mapping over lifelong or heuristic searches individually. These results demonstrate the versatility and computational benefits of lifelong planning, an underexploited technique that should be used more often in mobile robotics.

Acknowledgments

We thank William Halliburton for interfacing D* Lite to the hardware and performing the robot experiments. The Intelligent Decision-Making Group is partly supported by NSF awards to Sven Koenig under contracts IIS-9984827, IIS-0098807, and ITR/AP-0113881 as well as an IBM faculty partnership award. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations, agencies, companies or the U.S. government.

References

1. P. Agre and D. Chapman. Pengi: An implementation of a theory of activity. In *Proceedings of the National Conference on Artificial Intelligence*, pages 268–272, 1987.
2. S. Albers and M. Henzinger. Exploring unknown environments. *SIAM Journal on Computing*, 29(4):1164–1188, 2000.
3. R. De Almeida and C. Melin. Exploration of unknown environments by a mobile robot. *Intelligent Autonomous Systems*, 2:715–725, 1989.
4. R. Alterman. Adaptive planning. *Cognitive Science*, 12(3):393–421, 1988.
5. R. Arkin. Motor-schema based mobile robot navigation. *International Journal of Robotics Research*, 8(4):92–112, 1989.
6. R. Arkin and T. Balch. AuRA: Principles and practice in review. *Journal of Experimental and Theoretical Artificial Intelligence*, 9(2):175–189, 1997.
7. B. Awerbuch, M. Betke, R. Rivest, and M. Singh. Piecemeal graph exploration by a mobile robot. *Information and Computation*, 152(2):155 –172, 1999.
8. W. Burgard, D. Fox, M. Moors, R. Simmons, and S. Thrun. Collaborative multi-robot exploration. In *Proceedings of the International Conference on Robotics and Automation*, pages 476–481, 2000.
9. C. Choo, J. Smith, and N. Nasrabadi. An efficient terrain acquisition algorithm for a mobile robot. In *Proceedings of the International Conference on Robotics and Automation*, pages 306–311, 1991.

10. H. Choset. *Sensor-Based Motion Planning: The Hierarchical Generalized Voronoi Graph.* PhD thesis, California Institute of Technology, Pasadena (California), 1996.
11. X. Deng, T. Kameda, and C. Papadimitriou. How to learn an unknown environment; I: The rectilinear case. *Journal of the ACM*, 45(2):215 –245, 1998.
12. X. Deng and C. Papadimitriou. Exploring an unknown graph. In *Proceedings of the Symposium on Foundations of Computer Science*, pages 355–361, 1990.
13. G. Dudek, P. Freedman, and S. Hadjres. Using local information in a non-local way for mapping graph-like worlds. In *Proceedings of the International Conference on Artificial Intelligence*, pages 1639–1647, 1993.
14. G. Dudek, M. Jenkin, E. Milios, and D. Wilkes. Robotic exploration as graph construction. *IEEE Transactions on Robotics and Automation*, 7(6):859–865, 1991.
15. D. Frigioni, A. Marchetti-Spaccamela, and U. Nanni. Fully dynamic algorithms for maintaining shortest paths trees. *Journal of Algorithms*, 34(2):251–281, 2000.
16. A. Gerevini and I. Serina. Fast plan adaptation through planning graphs: Local and systematic search techniques. In *Proceedings of the International Conference on Artificial Intelligence Planning and Scheduling*, pages 112–121, 2000.
17. A. Goel, K. Ali, M. Donnelan, A. Gomez de Silva Garza, and T. Callantine. Multistrategy adaptive path planning. *IEEE Expert Journal*, 9(6):57–65, 1994.
18. K. Hammond. Explaining and repairing plans that fail. *Artificial Intelligence*, 45:173–228, 1990.
19. S. Hanks and D. Weld. A domain-independent algorithm for plan adaptation. *Journal of Artificial Intelligence Research*, 2:319–360, 1995.
20. S. Kambhampati and J. Hendler. A validation-structure-based theory of plan modification and reuse. *Artificial Intelligence*, 55:193–258, 1992.
21. J. Koehler. Flexible plan reuse in a formal framework. In Christer Bäckström and Erik Sandewall, editors, *Current Trends in AI Planning*, pages 171–184. IOS Press, 1994.
22. S. Koenig, D. Furcy, and Colin Bauer. Heuristic search-based replanning. In *Proceedings of the International Conference on Planning and Scheduling*, 2002.
23. S. Koenig and M. Likhachev. Improved fast replanning for robot navigation in unknown terrain. Technical Report GIT-COGSCI-2002/3, College of Computing, Georgia Institute of Technology, Atlanta (Georgia), 2001.
24. S. Koenig and M. Likhachev. Incremental A*. In *Advances in Neural Information Processing Systems 14*, 2001.
25. S. Koenig and M. Likhachev. Improved fast replanning for robot navigation in unknown terrain. In *Proceedings of the International Conference on Robotics and Automation*, 2002.
26. S. Koenig, C. Tovey, and W. Halliburton. Greedy mapping of terrain. In *Proceedings of the International Conference on Robotics and Automation*, pages 3594–3599, 2001.
27. B. Kuipers and Y. Byun. A robust, qualitative method for robot spatial learning. In *Proceedings of the National Conference on Artificial Intelligence*, pages 774–779, 1988.
28. S. Kwek. On a simple depth-first search strategy for exploring unknown graphs. In *Proceedings of the Workshop on Algorithms and Data Structures*, volume 1272 of *Lecture Notes in Computer Science*, pages 345 –353. Springer, 1997.
29. J. Leonard, H. Durrant-Whyte, and I. Cox. Dynamic map building for an autonomous mobile robot. *International Journal of Robotics Research*, 11(4):286–298, 1992.

30. Y. Liu, S. Koenig, and D. Furcy. Speeding up the calculation of heuristics for heuristic search-based planning. In *Proceedings of the National Conference on Artificial Intelligence*, 2002.
31. V. Lumelsky and A. Stepanov. Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape. *Algorithmica*, 2:403–430, 1987.
32. D. Mackenzie, R. Arkin, and J. Cameron. Multiagent mission specification and execution. *Autonomous Robots*, 4(1):29–57, 1997.
33. G. Oriolo, G. Ulivi, and M. Vendittelli. Real-time map building and navigation for autonomous robots in unknown environments. *IEEE Transactions on Systems, Man, and Cybernetics*, 28(3):316–333, 1998.
34. P. Panaite and A. Pelc. Exploring unknown undirected graphs. *Journal of Algorithms*, 33(2):281–295, 1999.
35. C. Papadimitriou and M. Yannakakis. Shortest paths without a map. *Theoretical Computer Science*, 84(1):127–150, 1991.
36. J. Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, 1985.
37. L. Prasad and S. Iyengar. A note on the combinatorial structure of the visibility graph in simple polygons. *Theoretical Computer Science*, 140(2):249–263, 1995.
38. G. Ramalingam and T. Reps. An incremental algorithm for a generalization of the shortest-path problem. *Journal of Algorithms*, 21:267–305, 1996.
39. N. Rao. Algorithmic framework for learned robot navigation in unknown terrains. *IEEE Computer*, 22(6):37–43, 1989.
40. N. Rao. Robot navigation in unknown generalized polygonal terrains using vision sensors. *IEEE Transactions on Systems, Man, and Cybernetics*, 25(6):947–962, 1995.
41. N. Rao, S. Hareti, W. Shi, and S. Iyengar. Robot navigation in unknown terrains: Introductory survey of non-heuristic algorithms. Technical Report ORNL/TM–12410, Oak Ridge National Laboratory, Oak Ridge (Tennessee), 1993.
42. L. Romero, E. Morales, and E. Sucar. An exploration and navigation approach for indoor mobile robots considering sensor's perceptual limitations. In *Proceedings of the International Conference on Robotics and Automation*, pages 3092–3097, 2001.
43. R. Simmons. A theory of debugging plans and interpretations. In *Proceedings of the National Conference on Artificial Intelligence*, pages 94–99, 1988.
44. K. Singh and K. Fujimura. Map making by cooperating mobile robots. In *Proceedings of the International Conference on Robotics and Automation*, pages 254–259, 1993.
45. A. Stentz. The focussed D* algorithm for real-time replanning. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1652–1659, 1995.
46. S. Thrun. Lifelong learning algorithms. In S. Thrun and L. Pratt, editors, *Learning To Learn*. Kluwer Academic Publishers, 1998.
47. S. Thrun, A. Bücken, W. Burgard, D. Fox, T. Fröhlinghaus, D. Hennig, T. Hofmann, M. Krell, and T. Schmidt. Map learning and high-speed navigation in RHINO. In D. Kortenkamp, R. Bonasso, and R. Murphy, editors, *Artificial Intelligence Based Mobile Robotics: Case Studies of Successful Robot Systems*, pages 21–52. MIT Press, 1998.
48. M. Veloso. *Planning and Learning by Analogical Reasoning*. Springer, 1994.

Learning How to Combine Sensory-Motor Modalities for a Robust Behavior

Benoit Morisset and Malik Ghallab

LAAS - CNRS, Toulouse
{bmorisse,malik}@laas.fr

Abstract. We are proposing here an approach and a system, called ROBEL, that enables a designer to specify and build a robot supervision system which learns from experience very robust ways of performing a task such as "navigate to". The designer specifies a collection of Hierarchical Tasks Networks (HTN) that are complex plans, called *modalities*, whose primitives are sensory-motor functions. Each modality is a possible combination these functions for achieving the task. The relationship between supervision states and the appropriate modality for pursuing a task is learned through experience as a Markov Decision Process (MDP) which provides a general policy for the task. This MDP is independent of the environment; it characterizes the robot abilities for the task.

1 Introduction

In robotics today, representations and techniques available for *task planning* are mostly effective at the abstract level of mission planning. Primitives for these plans are tasks such as "navigate to location5", "retrieve and pick-up object2". These tasks are far from being *primitive* sensory-motor functions. Their design is very complex. It is not much helped out by task planning techniques.[1]

Our purpose here is exactly the design of such tasks in a robust, generic way. We do claim that task planning can be helpful for this design. Certainly not as a collection of plug-and-play planners. But planning representations and techniques are useful for specifying alternative complex plans achieving a task. They are useful for learning a domain independent policy that chooses, in each supervision state, the best such a plan for pursuing the task.

The robots we are experimenting with are autonomous mobile platforms in structured environments (Figure 10). They are equipped with several sensors - sonar, laser, vision - and actuators, eventually with a robot arm. Our robot architecture has a *functional level*, for the sensory-motor functions, a *control level* [1], and a *decision level* for supervision and planning [2]. The architecture (Figure 1) relies on Genom [3] a development tool for the specification and integration of sensory-motor modules, and on Propice [4] a PRS-like environment for programming the supervision system.

[1] This may explain the weak interest of the robotics community in task planning.

M. Beetz et al. (Eds.): Plan-Based Control of Robotic Agents, LNAI 2466, pp. 157–178, 2002.

The functional level of our robots is fairly rich. It has several modules for the same function, e.g., for localization, for map building and updating, or for motion planning and control. These redundant modules are needed because of possible failures, and because no single method or sensor has a universal coverage. Each has its weak points and drawbacks. Robustness requires a diversity of means for achieving a sensory-motor function. Robustness also requires the capability to combine consistently several such functions into a modality and to choose among available modalities the most appropriate one for the current context. The ambition of ROBEL, the system proposed here, is to address this last capability.

ROBEL enables a designer to specify a supervision system which learns from experience very robust ways of performing a task such as "navigate to" (this is the task illustrated here on which we have extensively experimented). The designer specifies a collection of Hierarchical Tasks Networks (HTN) [5,6,7] that are complex plans, called *modalities*, whose primitives are sensory-motor functions. Each modality is a possible way of combining some of these functions to achieve the desired task. A modality has a rich context-dependent control structure. The relationship between supervision states and the appropriate modality for pursuing a task is far from being obvious. In ROBEL this relationship is learned through experience as a Markov Decision Process (MDP) [8] which provides a policy for achieving the task. This MDP characterizes the robot abilities for that task; it is independent of the environment although it may get improved through learning if the robot is moved to another environment.

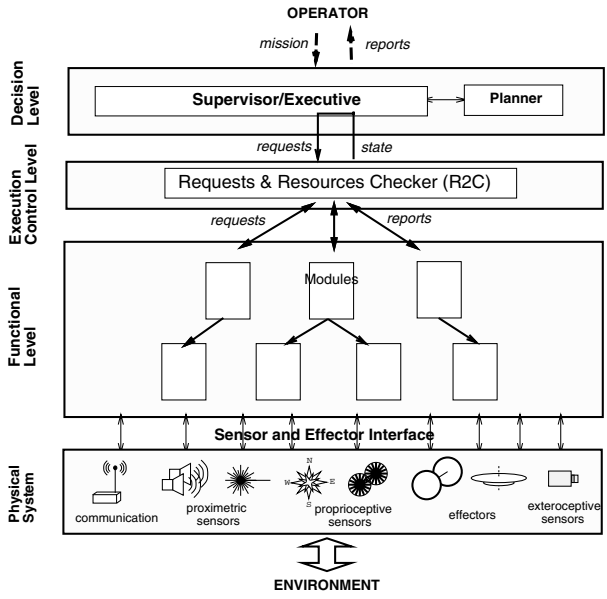


Fig. 1. The LAAS Architecture.

To summarize, primitives for ROBEL are redundant low-level sensory-motor *functions*. These functions, introduced in Section 3, are precisely modeled with their advantages and weak points; they are formalized as Genom modules [3]. Since the task at hand here is a navigation, the space representation is mostly relevant; it is introduced in Section 2. The HTN *modalities*, described in Section 4, enable to combine and control consistently a subset of functions in order to perform a task. Section 5 details the supervision system which chooses the appropriate modality for pursuing a task. We finally present ongoing experimental results (Section 6) and discuss the approach with respect to the state of the art and to future work.

2 Environment Representation

2.1 Metric Map

Most functions described here and the ROBEL supervision system itself rely on a model of the environment learned and maintained by the robot (see Section 3.1). The basic model is a 2D map of obstacle edges acquired by the laser (Figure 2). This map is generated through a supervised learning procedure which has to take place before autonomous navigation in the environment.

2.2 Topological Graph

A labeled topological graph of the environment is associated to the 2D map by hand specification in current implementation. Cells are polygon that partition the metric map. Each cell is characterized by its name and a *color* that corresponds to navigation features (Figure 3). We distinguish 7 different colors: Corridor, Corridor with landmarks, Large Door, Narrow Door, Confined Area, Open

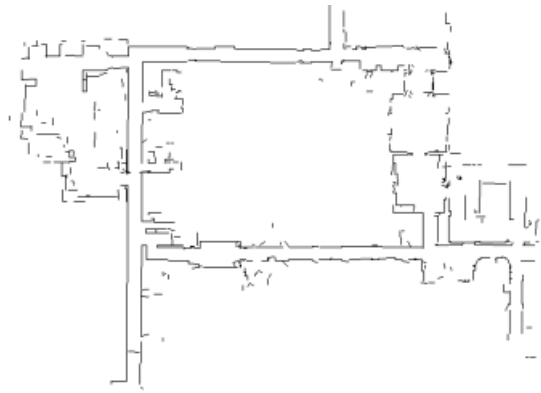


Fig. 2. The metric map (75m*50m)

Area, Open Area with fixed cameras. Edges of the topological graph are labeled by an estimate of the transition length from one cell to the next and by a heuristic estimate of how easy is such a transition.

3 Sensory-Motor Functions

In our architecture, sensory-motor functions are defined as a set of Genom modules [3]. A module may integrate several functions, each corresponding to a specific query to the module. A report is sent back by a module once the query is executed, indicating to the controller either the end of a nominal execution, or giving it additional information for non nominal cases. Some of the sensory-motor functions of the robot, needed later, are introduced below, indicating for each one their main non-nominal reports and the associated control.

3.1 Segment-Based Localization

This Simultaneous Map Building and Localization procedure uses an Extended Kalman Filter to match the local perception with the previously built model [9]. It relies on the 2D map of obstacle edges incrementally built by the robot from laser range data. It offers a continuous position updating mode, used when a good probabilistic estimate of the robot position is available. This function maintains an ellipsoid of uncertainty from the robot position variance, and it relies on a predicted local aspect of the environment.

When the robot is lost, a re-localization mode is available. A constraint relaxation on the ellipsoid of uncertainty extends the search space until a good matching with the map is found.

• Advantages: This function works in a tracking mode with a frequency of 1 Hz. It is generally reliable and robust to partial occlusions, and much more precise than odometry (Figure 4 compares the two estimates).

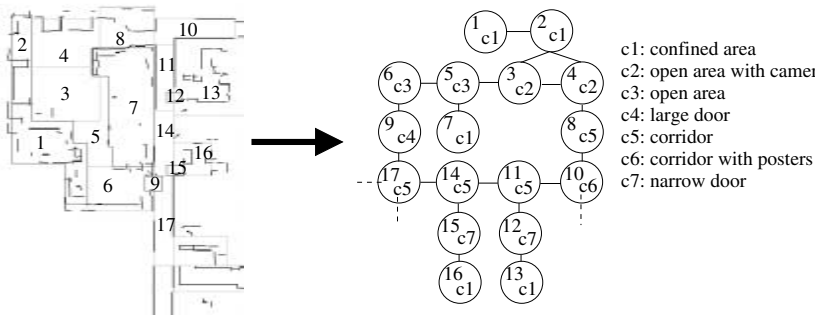


Fig. 3. Part of the topological map

Fig. 4. Segment-based localization and comparison to odometry

• Weaknesses: Laser occlusion gives unreliable data. This case occurs when dense unexpected obstacles are gathered in front of the robot. Moreover, in long corridors the laser obtains no data along the corridor axis. The perception of the right and left walls allows only a partial localization. Inaccuracy increases along the corridor axis. Restarting the position updating loop in a long corridor can prove to be difficult.
• Main associated control: A report of bad localization warns that the imprecision of the robot position has exceeded the allowed threshold. The robot stops, turns on the spot and re-activates the re-localization mode. This can be repeated in order to find a non-ambiguous corner in the environment to restart the localization loop.

3.2 Localization on Landmarks

This function uses monocular, grey level vision to detect known landmarks that are quadrangular, planar objects, e.g. doors or wall posters. It derives from the perceptual data an accurate estimation of the robot position [10,11]. This localization function works under the assumption of an indoor environment with vertical walls. It relies on a calibrated vision system.
• Advantages: This function gives a very accurate estimate of the robot position (about one centimeter). The setting up is simple. Few wall posters learned in long corridors allow locally an accurate localization.
• Weaknesses: Landmarks are available and visible in few areas of the environment. Further, in current implementation this function requires the robot stops. Hence the function is mainly used to correct from time to time the last robot position known.
• Main associated control: A report of a potentially visible landmark indicates that the robot enters an area of visibility of a landmark. The robot stops, turns towards the expected landmark; it searches it using the pan-tilt mount. A failure report notifies that the landmark was not identified. Eventually, the robot retries from a second predefined position.

3.3 Absolute Localization

A set of fixed calibrated cameras that cover a particular area is used to recognize and localize the robot [12]. A simple pattern on the robot simplify this recognition. A precise 3D localization is computed from the recognition pattern and from other fixed points in the scene. The accuracy improves with the number of robot displacements in the covered area. The inboard function relies on radio communication with the external vision system.

• Advantages: Very few occlusions occur in the vertical axis. The localization is very accurate and robust.

• Weaknesses: This function works only when the robot is within the covered area. Furthermore, there is a delay due to the interaction with the fixed vision system. The robot has to stop its motion each time it wants to know its absolute position in the covered area.

• Main associated control: A report of a recognition failure can be due to a robot position outside of the covered area or to other reasons such as insufficient light, occluded recognition pattern, etc.

3.4 Path Planner

This function plans a feasible path for the robot in a metrical model towards some goal. It is able to compute trajectories for holonomous as well as non-holonomous robots [13]. The space model used is a discretized bitmap derived from the learned map.

• Advantages: This path planner is fairly generic and robust.

• Weaknesses: The path has to be further processed to get an executable dynamic trajectory. Moreover, the planned path doesn't take into account environment change that occurs during navigation.

• Main associated control: A report may warn that a goal is too close to an obstacle. The planner computes a corrected goal position and a new path is computed. Another warning report is issued if the initial robot position is too close to an obstacle. The planner cannot produce a safe trajectory. The robot has to be moved away from the obstacles by a reactive motion function before a new path is queried.

3.5 Elastic Band for Plan Execution

This function updates and maintains dynamically a flexible trajectory as a sequence of configurations from the current robot position to the goal. Connexity between configurations relies on a set of internal forces that are used to optimize the global shape of the path. External forces are associated to obstacles and are applied to all configurations in order to dynamically update the path away from obstacles [14,15]. This function takes into account the laser data and the learned map.

• Advantages: A band is a series of robot configurations. Its format makes it very easy to couple to a planed path giving a very robust method for long range navigation.

• Weaknesses: A mobile obstacle can block the band against a static obstacle. The band may fall into local minima. The dynamic deformation is a costly step; this may limit the reactivity in certain cluttered, dynamic environments. This also limits the band length.
• Main associated control: A report may warn that the band execution is blocked by a temporary obstacle that cannot be avoided (e.g. a closed door, an obstacle in a corridor). This obstacle is perceived by the laser and is not represented in the map. If the band relies on a planed path, the new obstacle is added to the map. A new trajectory taking account the unexpected obstacle is computed, and a new elastic band is executed. Another report may warn that the actual band is no longer adapted to the planed path. It this case, a new band has to be created.

3.6 Reactive Obstacle Avoidance

This function provides a reactive motion capability. It works in two steps. First, it extracts from sensory data a description of free regions. It selects one such region, the closest to the goal location; it evaluates the robot safety based on the distance to obstacles; if possible it computes and achieves a motion command to that region [16] (Figure 5).
• Advantages: This method offers reactive motion capability that remains efficient in very cluttered space.
• Weaknesses: Like all the reactive methods, it may fall into local minima. Without a reference path, this method is not appropriate to perform long range navigation.
• Main associated control: A failure report is generated when the reactive execution is blocked.

Fig. 5. Reactive Obstacle Avoidance

4 Task Modalities

A navigation task (`Goto x y` θ) given by a user or by a mission planning step requires an integrated use of several functions among those presented earlier.

Each consistent combination of these functions gives a new behavior, a new way to perform the task, with its specific characteristics that make it more appropriate for some contexts or environments. Each such combination is a particular plan called a *modality*. We have specified and implemented 4 different and complementary modalities for the navigation task. Let us exemplify one such modality before giving the detail of the HTN representation for modalities and the associated control system.

4.1 Example of a Modality

Modality M_1 uses 3 functions: the path planner, the elastic band for the dynamic motion execution, and the laser-based localization. When M_1 is chosen to carry out a navigation, the laser-based localization is initialized. The robot position is maintained with a frequency of 1 Hz. A path is computed to reach the goal position. The path is carried out by the elastic band function. Stopping the modality interrupts the band execution and the localization loop; it restores the initial state of the map if temporary obstacles have been added to it. Suspending the modality stops the band execution. The path, the band, the localization loop are maintained. A suspended modality can be resumed by restarting the execution of the current elastic band.

4.2 Representation for Modalities

Modalities are represented as Hierarchical Task Networks [7]. The HTN formalism is adapted to our needs because of its expressiveness and its flexible control structure [5,6]. HTNs offer a middle ground between programming and automated planning, allowing the designer to express control structure when available, as in our case. Our HTNs for modalities are for the moment specified manually, but the synthesis of all modalities from generic specifications seems a feasible objective because of this representation.

Our HTNs are And/Or trees. An internal node is a task or a subtask that can be pursued in different context-dependent ways, which are the *Or-connectors*. Each such Or-connector is a possible decomposition of the task into a conjunction of subtasks. There are two types of *And-connectors*: with sequential or with parallel branches. Branches linked by a sequential And-connector are traversed sequentially in a depth-first manner. Branches linked by an parallel And-connector are traversed in parallel, in a breadth-first way. The leaves of the tree are primitive actions, each corresponding to a unique query addressed to a sensory-motor function. Thus, a root task is dynamically decomposed, according to the context, into a set of primitive actions organized as concurrent or sequential subsets. Execution starts as soon as the decomposition process reaches a leaf even if the entire decomposition process of the tree is not complete.

A primitive action can be *blocking* or *non-blocking*. In blocking mode, the control flow waits until the end of this action is reported before starting the next action in the sequence flow. In non-blocking mode actions in a sequence are triggered sequentially without waiting for a feedback. A blocking primitive action

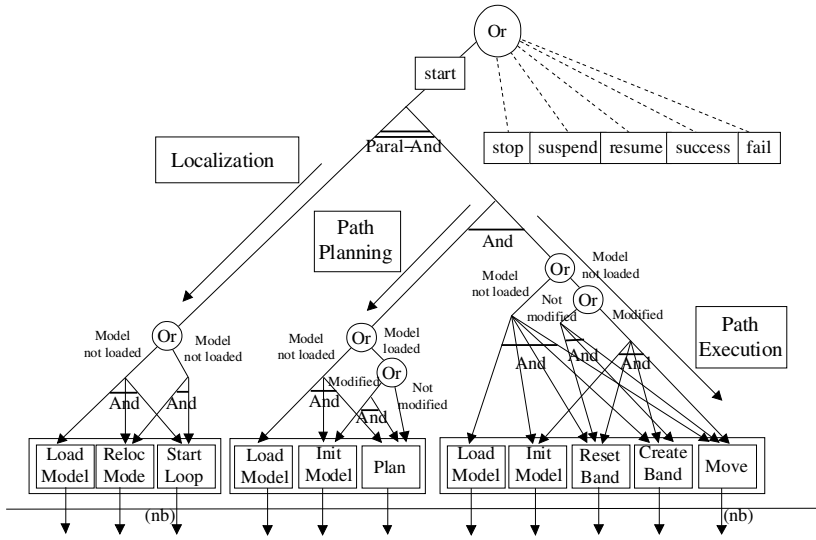


Fig. 6. Part of modality M_1

is considered ended after a report has been issued by the function and after that report has been processed by the control system. The report from non-blocking primitive action may occur and be processed after an unpredictable delay.

In our case, every modality tree (see Figure 6) starts with 6 Or-connectors labeled `start, stop, suspend, resume, succeed` and `fail`. The `start` connector represents the nominal modality execution, the `stop` connector the way to stop it and to restore the neutral state, characterized by the lack of any function execution. Furthermore, the environment model modified by the modality execution recovers its standard form. The `suspend` and `resume` connectors are triggered by the control system described below. The `suspend` connector allows to stop the execution by freezing the state of the functional level. The `resume` connector restarts the modality execution from such a frozen state. The `fail` (resp. `succeed`) connector is followed when the modality execution reaches a failure (resp. a success) end. These connectors are used to restore the neutral state and allow certain executions required in these specific cases.

4.3 Control and Resource Sharing

The feedback from sensory-motor functions to modalities has to be controlled as well as the resource sharing of parallel activities. The control system catches and react appropriately to reports emitted by functions. We found it convenient to also specify the control as HTNs (see Figure 7). Reports from functions play the same role in the control system as tasks in modalities. A report of some

type activates its own dedicated control HTN in a reactive way. A control tree represents a temporary behavior and cannot be interrupted . Each non nominal report points out a non nominal function execution. The aim of the corresponding control tree is to recover to a nominal modality execution. Some non nominal reports can be non recoverable failures. In these cases, the corresponding control sends a "fail" message to the modality pursuing this function. Nominal reports may notify the success of the global task. In this case, the "success" alternative of the modality is activated.

Resources to be managed are either physical non-sharable resources (e.g. motors, cameras, pan-tilt mount) or logical resources (the environment model that can be temporally modified). The execution of a set of concurrent non-blocking actions can imply the simultaneous execution of different functions. Because of that, several reports may appear at the same time, and induce the simultaneous activation of several control activities. These concurrent executions may generate a resource conflict. To manage this conflict, a resource manager organizes the resource sharing according to priorities:

• Each non-sharable resource is semaphorized. The request for a resource takes into account the priority level of each consumer. This priority level is specified by the designer.

• The execution of a control HTN is not interruptible. If another HTN requires a resource already in use by a control execution, the message activating this HTN (either modality or control) is added to a spooler according to its priority level.

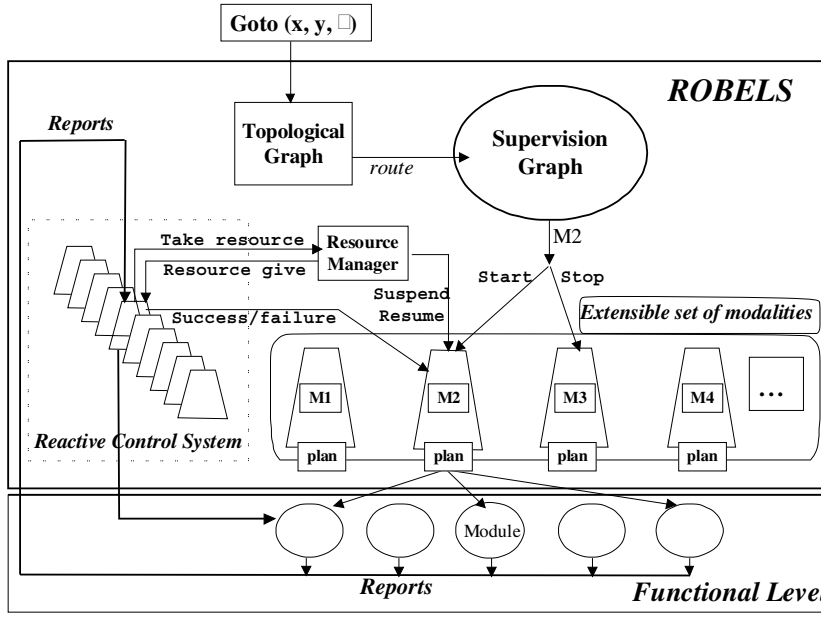


Fig. 7. The ROBEL supervision system

• A control HTN has a priority higher than those of start and suspend connectors but lower than those of stop and fail connectors.

When a non-nominal report is issued, a control HTN starts its execution. It requests the resource it needs. If this resource is already in use by a start connector of a modality, the manager sends to this modality a suspend message, and leaves a resume message for the modality in the spooler according to its priority. The suspend alternative is executed freeing the resource, enabling the control HTN to be executed. If the control execution succeeds, waiting messages are removed and executed until the spooler becomes empty. If the control execution fails, the resume message is removed from the spooler and the fail alternative is executed for the modality.

4.4 Other Navigation Modalities

We present here briefly four other modalities that have different execution conditions and rely on different functions.

Modality M_2 uses 3 functions: the path planner, the reactive obstacle avoidance and the laser-based localization. The path planner provides way-points (vertices of the trajectory) to the reactive motion function.

Despite these way-points the reactive motion can be trapped into local minima in highly cluttering environments. Its avoidance capability is higher than that of the elastic band function. However, the high reactivity to obstacles together with the attraction to way-points causes a more oscillating and discontinuous motion which confuses the localization function. This is a clear drawback for M_2 in the long corridors.

Modality M_3 is as M_2 but without path planning and with a reduced speed obstacle avoidance. The start connector starts the reactive motion and activates the laser-based localization loop.

Modality M_3 offers an efficient alternative in narrow environments like offices, and in cluttered spaces where planning can fail. It can be preferred to modality M_1 to avoid unreliable re-planning steps if the elastic band is blocked by a cluttered environment. Navigation is only reactive, hence with a local minima problem. The weakness of the laser localization in long corridors is also a drawback for M_3.

Modality M_4 uses the reactive obstacle avoidance function with the odometer and the visual landmark localization functions. The odometer inaccuracy can be locally reset by the visual localization function when the robot goes by a known landmark.

Reactive navigation between landmarks allows to cross a corridor without an accurate knowledge of the robot position. Typically this M_2 modality can be used in long corridors. The growing inaccuracy can make it difficult to find out the next landmark. The search method allows some inaccuracy on the robot position by moving the cameras but this inaccuracy cannot exceed one meter.

For this reason landmarks should not to be too far apart with respect to the required updating of odometry estimate. Furthermore, the reactive navigation of M_2 may fall into a local minima.

Modality M_5 relies on the reactive obstacle avoidance function and the absolute localization function with fixed cameras. The start connector starts the reactive motion and activates periodically the position processing. When the laser is occluded and when no landmark is visible, this last localization method is very efficient if the robot is within the area covered by fixed cameras.

5 The Supervision System

5.1 Supervision State-Variables

The supervision system has to choose a modality for pursuing a task which is most appropriate to current supervision state. In order to do this, *supervision state-variables* have to reflect control information of sensory-motor functions. These state-variables are the following:

• **Cluttering of the environment**. This is an important information to establish the execution conditions of the motion and localization functions. This state-variable is defined as a weighted sum of the distances to nearest obstacles perceived by the laser, with a dominant weight along the robot motion axis.

• **Angular variation of the profile**. Dots perceived by the laser are linked together to form a profile. The global angular variation of this profile is a state-variable that characterizes the environment in a complementary way. Close to a wall, the cluttering value is high but the angular variation remains low. On the other hand, in an open area, the cluttering is low while the angular variation may be high.

• **Precision of the position estimate.** The quality of the position estimate is computed from the co-variance matrix maintained by each localization function.

• **Confidence in the position estimate.** The inaccuracy is not sufficient to qualify the localization. Each localization function supplies a confidence estimate about the last processed position.

• **Properties of current area.** When the robot position estimate falls within some labeled cell of the topological graph, the corresponding color labels are taken into account, i.e., `Corridor`, `Corridor with Posters`, `Large Door`, `Narrow Door`, `Confined Area`, `Open Area`, `Open Area with Cameras`.

• **Modality in use.** This information is essential to assess the supervision state and possible transitions between modalities.

A supervision state is characterized by the values of these state-variables. In addition, we have a global failure state, that is reached whenever the control of a modality reports a failure.

5.2 Supervision Automata

Continuous supervision state-variables are discretized over few significant intervals. This leads to a discrete state-space which enables to define a *supervision automata*. This automata is non-deterministic: unpredictable external events may modify the environment, e.g. someone passing by may change the value of the cluttering variable, or the localization inaccuracy variable. Therefore the execution of the same modality in a given state may lead to different adjacent states.

This non-deterministic supervision automata is defined as the tuple $S_A = \{S, A, P, C\}$:

- S is a finite set of supervision states,
- A is a finite set of modalities, we denote $a(s)$ as the set of states that can be reached from s with the modality a,
- $P : S \times A \times S \rightarrow [0, 1]$ is a probability distribution on the state-transition function. We denote $P_a(s'|s)$ as the probability that the execution of modality a in state s leads to state s', $P_a(s'|s) \; /= 0$ iff $s' \in a(s)$,
- $C : A \times S \times S \rightarrow \Re^+$ is a positive cost function, $c(a, s, s')$ corresponds to the average cost (defined later) of performing the state transition from s to s' with to the modality a.

A and S are given by design from the definition of the set of modalities and of the supervision state-variables. We now have 5 modalities and about few thousands states. P and C are obtained from observed statistics during a learning phase (Section 5.5).

The supervision automata S_A is formally a Markov Decision Process. As an MDP, S_A could be used reactively on the basis of some universal policy π which selects for given state s the best modality $\pi(s)$ to be executed. The policy usually optimizes a general utility criterion that abstracts away the current navigation goal. We are proposing here another more precise approach that takes into account explicitly the navigation goal, transposed into S_A as a set S_g of supervision goal states. This set S_g is given by a look-ahead mechanism based on a search for a path in S_A that reflects a topological route to the navigation goal.

5.3 Finding a Set of Goal States

Given a navigation task, a search in the topological graph is achieved for an optimal route to the goal, taking into account estimated cost of edges between topological cells. This optimal route r is characterized by the pair (σ_r, l_r), where $\sigma_r = \langle c_1 c_2 \dots c_k \rangle$ is the sequence of colors of traversed cells, and l_r is the length of r.

Now, a path between two states in S_A defines also a sequence of colors σ_{path}, those of traversed states; it has a total cost, that is the sum $\sum_{path} C(a, s, s')$ over all traversed arcs. A path in S_A from the current supervision state s_0 to a state s corresponds to the planned route when the path *matches* the features of the route (σ_r, l_r) in the following way:

- $\sum_{path} c(a, s, s') \geq Kl_r$, K being a constant ratio from the cost of a state-transition to corresponding route length,
- σ_{path} corresponds to the same sequence of colors as σ_r with possible repetition factors, i.e., there are factors $i_1 > 0, \ldots, i_k > 0$ such that $\sigma_{path} = \langle c_1^{i_1} c_2^{i_2} \ldots c_k^{i_k} \rangle$ when $\sigma_r = \langle c_1 c_2 \ldots c_k \rangle$.

This last condition requires that we will be traversing in S_A supervision states having to the same color as the planned route. A repetition factor corresponds to the number of supervision states, at least one, required for traversing a topological cell. The first condition enables to prune paths in S_A that meet the condition on the sequence of colors but cannot correspond to the planned route. However, when such a path of a total cost smaller than Kl_r is found, it is first checked for a possible loop involving a repetition factor. If one is found then the cost condition is necessarily met; otherwise the path is pruned.

Let the predicate $\mathsf{route}(s_0, s)$ be true whenever the optimal path in S_A from s_0 to s meets the two previous conditions. By definition $s \in S_g$ iff $\mathsf{route}(s_0, s)$ holds.

A Moore-Dijkstra algorithm starting from s_0 gives optimal paths to all states in S_A in $O(n^2)$ (Figure 8). For every such a path, the predicate $\mathsf{route}(s_0, s)$ is checked in a straightforward way. In this algorithm, $f(s)$ is the cost of the current path from s_0 to s as defined by the backpointers $\mathsf{father}(s)$, $\mathsf{cost}(s, s')$ is simply the minimum of $C(a, s, s')$ over all modalities. Since Moore-Dijkstra explores paths of increasing length, an upper-bound on the total path cost enables a restricted and efficient search in S_A for goal states S_g.

```
Moore-Dijkstra(s_0, S, C)
    f(s_0) ← 0; ∀s ≠ s_0 : f(s) ← ∞
    P ← S; S_g ← ∅
    while P ≠ ∅ do
        remove from P the state ŝ with the minimal f
        if route(s_0, ŝ) then add ŝ into S_g
        foreach s' ∈ P adjacent to ŝ do
            if f(s') > f(ŝ)+cost(ŝ, s') then do
                f(s') ← f(ŝ)+cost(ŝ, s')
                father(s') ← s
return(S_g)
```

Fig. 8. Finding goal states

It is important to notice that this set S_g of supervision goal states is a *heuristic projection* of the planned route to the goal. There is no guaranty that following blindly (i.e., in an open-loop control) a path in S_A that meets $\mathsf{route}(s_0, s)$ will lead to the goal; and there is no guaranty that every successful navigation to the goal corresponds to a sequence of supervision states that meets $\mathsf{route}(s_0, s)$.

This only an efficient and reliable way of focusing the MDP cost function with respect to the navigation goal and to the planned route.

5.4 Finding a Control Policy

At this point we would like to find the best modality to apply to the current state s_0 in order to reach a state in S_g, given the probability distribution function P and the cost function C.

A simple adaptation of the classical *Value Iteration* algorithm solves this problem (Figure 9). In this algorithm $E(s)$ denotes the expected cost of reaching a goal state from s; $l(a)$ is a local variable for the expected cost in s with modality a; the output is a policy π, the modality to apply to to each state. The stopping criterion is: $max\{\Delta E(s) \mid s \in S\} < \epsilon$; $\Delta E(s)$ being the decrease of $E(s)$ along an update. In practice a small fixed number of iterations is sufficient.

In our case we only need to know $\pi(s_0)$. Hence the above algorithm can be focused on a subset of states, basically those explored by the Moore-Dijkstra algorithm. The closed-loop control uses this policy as follows:

- The computed modality $\pi(s_0)$ is executed;
- The robot observes the new current state, it updates its route r and its set S_g of goal states, it finds the best modality to apply to current state.

This is repeated until the control reports a success or a failure. Recovery from a failure state consists in trying from the parent state an untried modality. If none is available, a global failure of the task is reported.

5.5 Learning the Supervision Automata

A sequence of randomly generated navigation goals is given to the robot. During its motion, new supervision states are met and new transitions are recorded or updated. The supervision state is observed with a high frequency, 10 Hz in our

```
Value-Iteration(S, A, P, C, S_g)
    foreach s ∈ S do
        if s ∈ S_g then E(s) ← 0
        else E(s) ← ∞
    while the stopping criterion is not true do
        foreach s ∈ S do
            foreach a ∈ A do l(a) ← Σ_{s'∈a(s)} P_a(s'|s)[C(a, s, s') + E(s')]
            E(s) ← min_{a∈A} l(a)
            π(s) ← arg min_{a∈A} l(a)
return(π)
```

Fig. 9. Finding the control policy

implantation, i.e. 15cm at the highest robot speed. It is updated whenever a state-variable changes.

Each time a transition from s to s' with modality a is performed, the traversed distance and speed are recorded, and the average speed v of this transition is updated. The cost of the transition $C(a, s, s')$ is the weighted average of $\frac{\omega}{v}$ observed for this transition; ω is a weight that takes into account the eventual control steps required during the execution of the modality a in s together with the outcome of that control. The statistics on $a(s)$ are recorded to update the probability distribution function.

Several strategies can be defined to learn S_A, e.g.:

- A modality is chosen randomly for a given task; this modality is pursued until either it succeeds or a fatal failure is notified. In this case, a new modality is chosen randomly and is executed according to the same principle. This strategy is used initially to expand S_A.
- S_A is used according to the normal control except in a state on which not enough data has been recorded; a modality is randomly applied to this state in order to augment known statistics, e.g, the random choice of an untried modality in that state.

6 Experimental Results

Fig. 10. A cluttered environment

The five modalities described earlier have been fully implemented on the Diligent Robot [17,18] and extensively experimented with. In order to characterize the usefulness domain of each modality we measured in a series of navigation tasks, the success rate and other parameters such as the distance covered, the

average speed, the number of retries and control actions. Various types of navigation conditions have been considered with several degradations of navigation conditions obtained by:

- Modifying the environment: cluttering an area (Figure 10) and hiding landmarks
- Modifying navigation data: removing essential features from the 2D map

Fig. 11. A complete occlusion of 2D edges

Five different types of experimental conditions have been distinguished:

- Case 1: Very long range navigation in nominal environment and map, e.g., turning around a circuit of 250 meters of corridors several times
- Case 2: Traversal of a highly cluttered area (as in Figure 10);
- Case 3: Traversal of the area covered by fixed cameras with the occlusion of the 2D characteristic edges of that area (Figure 11).
- Case 4: Traversal of very long corridors
- Case 5: Traversal of long corridors with hidden landmarks and a map degraded by removal of several essential 2D features, such as the contour of a very characteristic heating appliance (Figure 12)

The experimental results are summarized in table 1, where $\#r, d$, and v denote respectively the number of runs, the distance covered and the average velocity of the total navigation task, SR is the success rate of the experiment. In some cases, the performance of a modality varies widely depending on the specifics of each run. Instead of averaging out the results, we found it more meaningful to record and analyze the variation conditions. The following comments clarify the experimental conditions and some results:

Fig. 12. Degradation of a long corridor

- Modality M_1: SR drops sharply in case 1 if the circuit contains a narrow pass of less than 1m with sharp angles, such as a door in a 90 degree angular corridor; similarly for case 2 with the cluttering of the environment where the results range from easy success to complete failure; as expected M_1 fails completely in case 3 if the 2D edges are occluded; in case 4 and 5 longer corridors and/or imprecise initial localization reduces significantly SR.
- Modality M_2: is significantly more robust in case 1 to narrow and intricate passes, however it leads to less precise laser localization which explains lower values of SR in all cases; as for M_1, a wide spectrum of results is obtained in case 2 depending on the environment conditions; for cases 4 and 5 the same remark applies to M_1 and M_2.
- Modality M_3: requires for case 1 several way-points that may lead to success or to failure in local minima; it is very successful in case 2 even when the distance between obstacles are as low as 0.75m, but it requires a slow speed; case 3 has not been tested because M_3, as M_1, fails predictably; but it should succeed in case 4 that has no local minima and hence offers no difficulty for M_3; the performance for case 5 is similar to that of M_1.
- Modality M_4: good results have been recorded in cases 1 and 2 when the visual landmarks are dense, about 5m apart, otherwise SR drops sharply; this is particularly illustrated in case 3; for cases 4 and 5 M_4 is not sensitive to the length of the corridor and the degradation of its map as long as it contains enough landmarks.
- Modality M_5 (not shown in the table): fails everywhere but in case 3 where it has a performance similar to that of M_3.

An interesting view of our results is that for each case there is at least one successful modality. These are M_1 or M_2 for case 1, M_2 for case 2, M_3 or M_5 for

Table 1. Experimental results

Modality	case 1	case 2	case 3	case 4	case 5
M_1	$\#r = 20$ $SR = 100\%$ $d = 2320m$ $v = 0.26m/s$	$\#r = 5$	$\#r = 5$ $SR = 0\%$	$\#r = 20$ $SR = 100\%$	$\#r = 20$ $SR = 5\%$
M_2	$\#r = 12$ $SR = 80\%$ $d = 870m$ $v = 0.28m/s$	$\#r = 5$	$\#r = 5$ $SR = 0\%$	$\#r = 12$ $SR = 80\%$	$\#r = 0$ $SR \leq 5\%$
M_3	$\#r = 10$	$\#r = 10$ $SR = 100\%$ $v = 0.14m/s$	$\#r = 5$ $SR = 0\%$	$\#r = 0$	$\#r = 0$ $SR \leq 5\%$
M_4	$\#r = 0$	$\#r = 0$	$\#r = 0$	$\#r = 20$ $SR = 95\%$	$\#r = 12$ $SR = 100\%$ $v = 0.15m/s$

case 3, M_4 for cases 4 and 5. This clearly supports the approach of a supervision controller that switches from one modality to another one according to the context.

The above results correspond to navigation with a single modality. Four modalities, M_1, M_2, M_3 and M_4, have been demonstrated together [10] using a set of selection rules specified manually. The development of these rules triggered our work on learning a robust supervision system. We are achieving the implementation of the MDP-based supervision automata with the search method coupled to the topological graph and with the learning procedures. We started generating the S_A automata. It has a reasonable size of about few thousands states. We plan to test how portable is the learned supervision automata by bringing the robot to a new environment and by studying its behavior and the evolution of S_A. Another significant test will be the incremental addition of a new modality.

7 Discussion and Conclusion

This paper addressed the issue of robust supervision for task executing in an autonomous robot. We believe to have brought here two contributions. We have introduced a representation based on HTNs which enables to combine various sensory-motor functions in order to specify complex behaviors as a modalities that have very flexible control structures. We have developed in Propice, a PRS-like environment for programming reactive controllers, 5 such modalities devoted to navigation tasks.

This is certainly not the first contribution that relies on a planning formalism and on plan-based control in order to program an autonomous robot. For example, the *"Structured Reactive Controllers"* [19] are close to our concerns and

have been demonstrated effectively on the Rhino mobile robot. The efforts for extending and adapting the Golog language [20] to programming autonomous robots offer another interesting example from a quite different perspective, that of the Situation Calculus formalism [21]. The *"societal agent theory"* of [22] offers also another interesting approach for specifying and combining sequentially, concurrently or in a cooperative mode several agent-based behaviors; the CDL language used for specifying the agent interactions is similar to our Propice programming environment. Let us mention also the *"Dual dynamics"* approach of [23] that permit the flexible interaction and supervision of several behaviors. These are typical examples of a rich state of the art on possible architectures for designing autonomous robots (see [2] for a more comprehensive survey). Our particular and, to our knowledge, original contribution for the specification of behaviors or modalities relies in the formal use of HTNs with a dynamic expansion and traversal of task networks. The approach is effective because of the position of this representation between programming and automated planning. It seems indeed feasible to synthesize modalities from generic specifications. Furthermore, we have stressed here the need for, and we have studied the consistent use of several redundant HTN modalities to achieve robustness.

The second contribution of this work is an original approach for learning from the robot experiences an MDP-based supervision automata which enables to choose dynamically a modality appropriate to the current context for pursuing the task.

Here also the use of MDPs for supervision and control of robot navigation tasks is not new. Several authors expressed directly Markov states as cells of a navigation grid and addressed navigation through MDP algorithms, e.g. value iteration [24,25,26]. Learning systems have been developed in this framework. For example, XFRMLEARN extends these approaches further with a knowledge-based learning mechanism that adds subplans from experience to improve navigation performances [27]. Other approaches considered learning at very specific levels, e.g., to improve path planning capabilities [28]. Our approach stands at a more abstract and generic level. It addresses another purpose: acquiring autonomously the relationship from the set of supervision states to that of redundant modalities. We have proposed a convenient abstract supervision space, whose size, to some extent, can be controlled. We have also introduced a new and effective search mechanism that projects a topological route into the supervision automata. The learning of this automata relies on simple and effective techniques. As long as this abstract space has a moderate size of few thousands states the data acquisition process remains within a reasonable cost. The learned MDP is independent of a particular environment and characterizes the robot capabilities.

In addition to future work directions mentioned above, an important test of ROBEL will be the extension of the set of tasks to manipulation tasks such as *"open a door"*. This significant development will require the integration of new manipulation functions, the design of redundant behaviors for these tasks and their associated control, and the extension of the supervision state. We believe ROBEL to be generic enough to support and permit such developments.

References

1. F. Ingrand and F. Py. Online execution control checking for autonomous systems. In *IAS-7,Marina del Rey,California,USA*, 2002.
2. R. Alami, R. Chatila, S. Fleury, M. Ghallab, and F. Ingrand. An Architecture for Autonomy. *International Journal of Robotics Research*, 17(4):315–337, April 1998.
3. S. Fleury, M. Herrb, and R. Chatila. Genom: A tool for the specification and the implementation of operating modules in a distributed robot architecture. In *IROS, Grenoble, France*, volume 2, pages 842–848, September 1997.
4. F. F. Ingrand, R. Chatila, R. Alami, and F. Robert. PRS: A High Level Supervision and Control Language for Autonomous Mobile Robots. In *IEEE ICRA, St Paul, (USA)*, 1996.
5. K. Erol, J. Hendler, and D.S. Nau. HTN planning: Complexity and expressivity. In *AAAI*, 1994.
6. K. Erol, J. Hendler, and D.S. Nau. Complexity results for hierarchical task-network planning. In *Annals of Mathematics and AI*, pages 18:69–93, 1996.
7. E. Sacerdoti. A structure for plans and behavior. In *American Elsevier Publishing*, 1977.
8. D.J. White. Markov decision processes. In *John Wiley and Sons*, 1993.
9. P. Moutarlier and R. G. Chatila. Stochastic Multisensory Data Fusion for Mobile Robot Location and Environment Modelling. In *Proc. International Symposium on Robotics Research, Tokyo*, 1989.
10. J.B. Hayet, F. Lerasle, and M. Devy. Planar landmarks to localize a mobile robot. In *SIRS 2000, Berkshire, England*, pages 163–169, July 2000.
11. V. Ayala, J.B. Hayet, F. Lerasle, and M. Devy. Visual localization of a mobile robot in indoor environments using planar landmarks. In *IEEE IRS'2000, Takamatsu, Japan*, pages 275–280, November 2000.
12. S. Fleury, T. Baron, and M. Herrb. Monocular localization of a mobile robot. In *IAS-3, Pittsburgh, USA*, 1994.
13. J.P. Laumond, P.E. Jacobs, M. Taix, and R.M. Murray. A motion planner for nonholonomic mobile robots. *IEEE Transactions on Robotics and Automation*, 10(5):577–593, 1994.
14. S. Quinlan and O. Khatib. Towards real-time execution of motion tasks. In R. Chatila and G. Hirzinger, editors, *Experimental Robotics 2*. Springer Verlag, 1992.
15. M. Khatib. *Contrôle du mouvement d'un robot mobile par retour sensoriel*. PhD thesis, Université Paul Sabatier, Toulouse, December 1996.
16. J. Minguez and L. Montano. Nearness diagram navigation (ND): A new real time collision avoidance approach. In *IROS Takamatsu, Japan*, pages 2094–2100, 2000.
17. R. Alami, R. Chatila, S. Fleury, M. Herrb, F. Ingrand, M. Khatib, B. Morisset, P. Moutarlier, and T. Siméon. Around the lab in 40 days... In *IEEE ICRA, San Francisco, (USA)*, 2000.
18. R. Alami, I. Belousov, S. Fleury, M. Herrb, F. Ingrand, J. Minguez, and B. Morisset. Diligent: Towards a human-friendly navigation system. In *IROS 2000,Takamatsu, Japan*, 2000.
19. M. Beetz. Structured reactive controllers - A computational model of everyday activity. In *3rd Int. Conf. on Autonomous Agents*, pages 228–235, 1999.
20. H. Levesque et al. Golog: A logic programming language for dynamic domains. In *J. of Logic Programming*, pages 31:59–84, 1997.

21. R. Reiter. Natural actions, concurrency, and continuous time in the situation calculus. In *KR*, pages 2–13, 1996.
22. D.C. MacKenzie, R.C. Arkin, and J.M. Cameron. Multiagent mission specification and execution. In *Autonomous Robots, 4(1):29 V52*, 1997.
23. J. Hertzberg, H. Jaeger, Ph. Morignot, and U.R. Zimmer. A framework for plan execution in behavior-based robots. In *ISIC-98 Gaithersburg MD*, pages 8–13.
24. S. Thrun et al. Map-learning and high speed navigation in rhino. In ai-based mobile robots: case studies of successful robot systems. In *(Eds.), MIT Press,*, 1998.
25. L. Kaelbling et al. Acting under uncertainty: Discrete bayesian models for mobile-robot navigation. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 1996.
26. T. Dean and M. Wellman. Planning and control. In *Morgan Kaufmann*, 1991.
27. M. Beetz and T. Belker. Environment and task adaptation for robotics agents. In *ECAI*, 2000.
28. K.Z. Haigh and M. Veloso. Learningsituation-dependent costs: Improving planning from probabilistic robot execution. In *In 2nd Int. Conf.on Autonomous Agents*, 1998.

Execution-Time Plan Management for a Cognitive Orthotic System

Martha E. Pollack[1], Colleen E. McCarthy[2],
Sailesh Ramakrishnan[1], and Ioannis Tsamardinos[3]

[1] Artificial Intelligence Laboratory
Dept. of Electrical Engineering and Computer Science
University of Michigan, Ann Arbor, MI 48109, U.S.A
{pollackm,sailes}h@umich.edu
http://www.eecs.umich.edu/~pollackm/
[2] Department of Computer Science, University of Pittsburgh, U.S.A
colleen@cs.pitt.edu
[3] Department of Biomedical Informatics, Vanderbilt University, U.S.A
ioannis.tsamardinos@vanderbilt.edu

Abstract. In this paper we discuss our work on plan management in the Autominder cognitive orthotic system. Autominder is being designed as part of an initiative on the development of robotic assistants for the elderly. Autominder stores and updates user plans, tracks their execution via input from robot sensors, and provides carefully chosen and timed reminders of the activities to be performed. It will eventually also learn the typical behavior of the user with regard to the execution of these plans. A central component of Autominder is its Plan Manager (PM), which is responsible for the temporal reasoning involved in updating plans and tracking their execution. The PM models plan update problems as disjunctive temporal problems (DTPs) and uses the Epilitis DTP-solving system to handle them. We describe the plan representations and algorithms used by the Plan Manager, and briefly discuss its connections with the rest of the system.

1 Introduction

In this paper we discuss our work on plan management in the Autominder cognitive orthotic system. Autominder is being designed as part of the Initiative on Personal Robotic Assistants for the Elderly (Nursebot)[15], a multi-university collaborative project aimed at investigations of robotic technology for the elderly. The initial focus of this initiative is the design of an autonomous robot, currently called Pearl, that will "live" in the home of an elderly person. Autominder stores and updates plans representing the activities that the elderly client is expected to perform, tracks their execution via sensor input from the robot, learns the typical behavior of the client with regard to the execution of these plans, and provides carefully chosen and timed reminders for the activities to be performed.[1]

[1] The learning component is not yet implemented.

M. Beetz et al. (Eds.): Plan-Based Control of Robotic Agents, LNAI 2466, pp. 179–192, 2002.

A central component of Autominder, is its Plan Manager (PM), which is responsible for the temporal reasoning involved in updating plans and tracking their execution. The PM must be able to reason about complex temporal constraints. For example, the client's plan may specify constraints on when she should eat meals, e.g., that they should be spaced four hours apart. Thus, when the client wakes up and eats her first meal, Autominder must propagate this information to update the times for lunch and dinner. Adjustments may also need to be made to the timing constraints on other activities, such as taking medicine. Notice that the adjustments in the plan may be rather complex: for instance, if the new scheduled time for lunch conflicts with a recreational activity, such as a visit to a neighbor, it may be necessary to reschedule the visit.

Autominder's PM extends our earlier work in which we developed a prototype plan manager for an intelligent personal calendaring tool, called PMA (the Plan Management Agent) [20]. The goal of PMA is to allow a user to manage a complex schedule of tasks by checking whether new tasks conflict with existing ones, suggesting ways of resolving conflicts that are detected, and providing reminders to the user at the time at which an activity must be executed. The PM in Autominder extends PMA in two ways:

- It allows more expressive plan representations, notably supporting arbitrary disjunctive temporal constraints; and
- It provides efficient algorithms for handling plans with this level of expressiveness. In fact, the algorithms have been shown to be two orders of magnitude faster on a range of benchmark problems [24].

Additionally, we have added an execution monitor to Autominder that was not present in PMA, and we have separated out the reminding task into a distinct module that performs more detailed reasoning about the appropriateness of alternative reminders.

This paper focuses on Autominder's plan manager. In the next section, we describe our plan representation, showing how it supports richer set of temporal constraints between activities than most alternative representations. Section 3 describes the algorithms used by the plan manager to perform plan update at execution time, while Section 4 discusses the underlying reasoning system, Epilitis. Section 5 relates the plan manager to the larger Autominder system and Section 6 briefly describes the robot platform (Nursebot) that Autominder is designed to run on. Finally, the remainder of the paper discusses related and future work.

2 Plan Representation

The PM uses a library of plans that represent the structure of activities that the client typically performs. In our current Autominder domain these activities include taking medicine correctly, eating, drinking water, toileting, taking care of personal hygiene, performing physical exercises (e.g., "Kegel" bladder exercises),

performing self-examinations (e.g., foot exams by diabetics), engaging in recreational activities (e.g., watching television, attending a Bingo game), and going to doctors' and dentists' appointments. Many of these activities are modeled as being decomposable into more primitive activities. For instance, a doctor's appointment consists of making the appointment beforehand, arranging transportation, confirming the visit before leaving, going to the doctor's office, and scheduling a follow-up visit. Essentially, each high-level action corresponds to a complete partial-order plan with steps, causal links, and temporal constraints; however, as we describe below, the set of constraints we allow is much richer than that of most partial order planners.

At initialization the caregiver supplies Autominder with a typical daily plan of activities. Information need only be provided about "top-level" activities, e.g., a doctor's appointment; the entire plan for that activity, including all the steps (e.g., arranging transportation for the appointment) and constraints, is then loaded from the plan library and inserted into the client's overall daily plan. Two points should be made about plan initialization. First, while the plan library may provide default temporal constraints, the user has complete control over these, and can specify start times and durations for all activities. Second, additional activities can later be added, either as a one-time event (e.g., a visit from a relative) or a recurrent activity (e.g., attending a weekly Bingo game). The plan developed at initialization acts as a template for daily activities, not an absolute schedule.

The plan manager supports a rich set of temporal constraints: for example, we can express that the client should take a medication within 15 minutes of waking, and then eat breakfast between 1 and 2 hours later. Importantly, as indicated above, the time constraints can be flexible: fixed, rigid times do *not* need to be assigned to each activity. Instead, the plan may specify that an activity must be performed without specifying the exact time at which it should occur, or that a particular goal must be achieved without specifying what plan the client should use to achieve that goal. Figure 1 shows the types of temporal constraints that can be specified.

1	Earliest Start Time
2	Latest Start Time
3	Earliest End Time
4	Latest End Time
5	Minimum Duration
6	Maximum Duration
7	Minimum Period of Separation between Activities
8	Maximum Period of Separation between Activities

Fig. 1. Temporal Constraints for an Activity

To achieve this flexibility, we model client plans as Disjunctive Temporal Problems (DTP) [17,22,24]. A DTP is an expressive framework for temporal rea-

soning problems that extends the well-known Simple Temporal Problem (STP) [6] by allowing disjunctions, and the Temporal Constraint Satisfaction Problem (TCSP) [*ibid.*] by removing restrictions on the allowable disjunctions. DTPs are represented as a set of variables and a set of disjunctive constraints between members of this set. Formally, a DTP is defined to be a pair $<V, C>$, where

- V is a set of variables (or nodes) whose domains are the real numbers, and
- C is a set of disjunctive constraints of the form:
  $C_i : x_1 - y_1 \leq b_1 \vee \ldots \vee x_n - y_n \leq b_n$, such that x_i are y_i are both members of V, and b_i is a real number.[2]

In the PM, we assign a pair of DTP variables to each activity in the client's plan: one variable represents the start time of the activity, while the other represents its end time. We can easily encode a variety of typical planning constraints, including absolute times of events, relative times of events, and event durations, and can also express ranges for each of these. Figure 2 shows some of these relations between two activities, s_i and s_j. A simple example is shown in Figure 3 where we represent information about two activities, toileting and watching TV. Note that to express a clock-time constraint, e.g., TV watching beginning at 8:00am, we use a *temporal reference point* (TR), a distinguished value representing some fixed clock time. In Autominder the TR corresponds to midnight.

To express:	Use:
s_i precedes s_j	end(s_i) - start(s_j) $\leq$ 0
s_i begins at 9am, Monday	ref - start(s_i) $\leq$ -9 $\wedge$ start(s_i) - ref $\leq$ 9 (assuming ref is 12am on Monday)
s_i lasts between 2 and 3 hours	end(s_i) - start(s_i) $\leq$ 3 $\wedge$ start(s_i) - end(s_i) $\leq$ -2
s_i occurs more than 48 hours before s_j	end(s_i) - start(s_j) $\leq$ -48

Fig. 2. Temporal constraint representations

3 Plan Updates

The primary job of the Plan Manager is to maintain an up-to-date model of the plan activities that the user should execute. Updates occur in response to four types of events:

The addition of a new activity to the plan. During the course of the day, the client and/or his or her caregivers may want to make additions to the plan: for instance, to attend a Bingo game or a newly scheduled doctor's

[2] As is customary in the literature, in this paper we will assume without loss of generality that b_i is an integer.

"Toileting should begin between 11:00 and 11:15."
$660 \leq Toileting_S - TR \leq 675$
"Toileting takes between 1 and 3 minutes."
$1 \leq Toileting_E - Toileting_S \leq 3$
"Watching the TV news can begin at 8:00 or 11:00."
$480 \leq WatchNews_S - TR \leq 482 \vee$
$660 \leq WatchNews_S - TR \leq 662$
"The news takes exactly 30 minutes."
$30 \leq WatchNews_E - WatchNews_S \leq 30$
"Toileting and watching the news cannot overlap."
$0 \leq WatchNews_S - Toileting_E \leq \infty \vee$
$0 \leq Toileting_S - WatchNews_E \leq \infty$

Fig. 3. Examples of the use of DTP Constraints

appointment. At this point, plan merging must be performed to ensure that the constraints on the new activity are consistent with the other constraints in the existing plan.

The modification or deletion of an activity in the plan. Plan merging must also occur in the case of activity modification or deletion. Note that the PM will add or tighten constraints if needed, but will not "roll back" (i.e., weaken) any constraints. For instance, if the bounds on eating lunch are constrained to allow for an appointment, but the appointment is later retracted, the bounds on lunch will not be changed to allow for more time. More sophisticated plan retraction is an area of future research.

The execution of an activity in the plan. It is important for the PM to respond to the execution of activities in the plan. Information about activity execution is provided by another component of Autominder, the Client Modeler (CM). The CM is tasked with monitoring plan execution. It receives reports of the robot's sensor readings, for instance when the client moves from one room to another, and uses that to infer the probability that particular steps in the client plan have been executed; it can also issue questions to the client for confirmation about whether a step has been executed. When the CM believes with probability exceeding some threshold that a given step has begun or ended, it passes this information on to the PM. The PM can then update the client plan accordingly.

The passage of a time boundary in the plan. Finally, just as the execution of a plan step may necessitate plan update, so may the non-execution of a plan step. Consider our example in Figure 3, where the client wants to watch the news on television each day, either from 8:00-8:30 a.m. or from 11:00-11:30 p.m. At 8:00am (or a few minutes after), if the client has not begun watching the news, then the PM should update the plan to ensure that the 11:00-11:30 slot is reserved for that purpose.

To perform plan update in each of these cases, the PM formulates and solves a disjunctive temporal problem (DTP). Detailed explanations of the updates

performed in each of these four cases can be found in [19]. Here we just sketch the approach taken for the first case, when a new activity is added to the plan. In that case, the PM performs *plan merging*, a process that can be decomposed into the following steps:

1. creating DTP encodings of the existing plan and the new activity;
2. identifying potential conflicts introduced by the plan change;
3. creating a DTP that includes all the possible alternative resolutions of the conflicts; and finally
4. attempting to solve a DTP that combines the encodings in (1) and in (3).

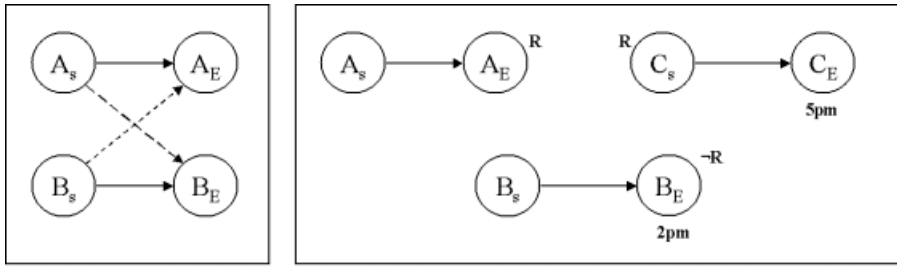


Fig. 4. Resolving temporal conflicts

Because we are using the DTP representation, we can model the traditional conflict resolution techniques (promotion and demotion) with a single constraint, as illustrated in Figure 4a: either A must be before B or B must be before A. But the use of DTPs also allows us to handle quantitative temporal constraints. In Figure 4b we illustrate an action A that achieves condition R for action C; we also include another action B that threatens the causal link from A to C. The DTP constraint that captures the alternative resolutions of this threat is $B_E - A_S \leq 0 \vee C_E - B_S \leq 0$. However, suppose further that action B must end by 2pm and action C by 5pm. It is then clear that only the first disjunct of the threat resolution constraint is satisfiable, and this is precisely what a DTP-solver would find. While this is a very simple example, in general the PM handles large sets of complex disjunctive constraints.

Clearly, solving DTPs is a central part of what the PM does. In the PM, we use the using the Epilitis DTP solving system developed in our group [24,25]. In the next section, we briefly describe DTP solving in general, and Epilitis in particular.

4 Solving DTPs

As stated earlier, a DTP is an extension of a Simple Temporal Problem (STP) [6]. Essentially, an STP is a DTP in which constraints must be simple inequalities

without any disjunctions; i.e., each constraint in C takes the form $x - y \leq b_{xy}$. This constraint represents the fact that y must occur no more than b_{xy} time units after x. Because an STP contains only binary constraints, it can be represented with a weighted graph called a Simple Temporal Network (STN), in which an edge (x, y) with weight b exists between two nodes iff there is a constraint $(y - x \leq b_{xy}) \in C$. Polynomial time algorithms can be used to compute the all-pairs shortest path matrix, or *distance array* of the STN. A path p from node x to node y imposes the following (induced) constraint: $y - x \leq \sum_{i=0}^{n} b_{p_i p_{i+1}}$ where $x = p_0$, $y = p_{n+1}$ and the rest p_i are the nodes on the path p. The tightest (induced or explicit) constraint between any nodes x and y is therefore given by the shortest path connecting them, denoted by d_{xy}, and called the *distance* between x and y. The *distance array* for a particular STN is its all-pairs shortest-path matrix. An STN is consistent if and only if for every node $x, d_{xx} \geq 0$, which means that there are no negative cycles, something that can be computed in polynomial time. Finding a solution to an STN is also a polynomial time computation.

A DTP can be viewed as encoding a collection of alternative STPs. To see this, recall that each constraint in a DTP is a disjunction of one or more STP-style inequalities. Let C_{ij} be the j^{th} disjunct of the i^{th} constraint of the DTP. If we select one disjunct C_{ij} from each constraint C_i, the set of selected disjuncts forms an STP, which we will call a *component STP* of a given DTP. It is easy see that a DTP D is consistent if and only if it contains at least one consistent component STP. Moreover, any solution to a consistent component STP of D is also a solution to D itself. Because only polynomial time is required both to check the consistency of an STP, and, if consistent, extract a solution to it, in the remainder of this paper we will say that the solution of a given DTP is any consistent component STP of it.

The computational complexity in DTP solving derives from fact that there are exponentially many sets of selected disjuncts that may need to be considered; the challenge is to find ways to efficiently explore the space of disjunct combinations. This has been done by casting the disjunct selection problem as a constraint satisfaction processing (CSP) problem [22,17] or a satisfiability (SAT) problem [1]. Because the original DTP is itself a CSP problem, we will refer to the component-STP extraction problem as the *meta-CSP* problem. The meta-CSP contains one variable for each constraint C_i in the DTP. The domain of C_i is the set of disjuncts in the original constraint C_i. The constraints in the meta-CSP are not given explicitly, but must be inferred: an assignment satisfies the meta-CSP constraints iff the assignment corresponds to a consistent component STP. For instance, if the variable C_i is assigned the value $x - y \leq 5$, it would be inconsistent to extend that assignment so that some other variable C_j is assigned the value $y - x \leq -6$.

In Autominder, we use the Epilitis DTP solver developed in our research group[24,25]. Like prior DTP solvers [1,17,21], Epilitis does not attempt to solve the DTP directly by searching for an assignment of integers to the time points. Instead, it solves a meta-CSP problem: it attempts to find one disjunct from each disjunctive constraint such that the set of all selected disjuncts forms a

consistent STP. Epilitis can return an entire STP, which provides interval rather than exact constraints on the time points in the plan. Consider an example of a plan that involves taking medicine between 14:00 and 15:00, which is amended with a plan to leave for a bridge game at 14:30. Epilitis will return a DTP that constrains the medicine to be taken sometime between 14:00 and 14:30; it does not have to assign a specific time (e.g., 14:10) to that action.

In general, there may be multiple solutions to a DTP, i.e., multiple consistent STPs that can be extracted from the DTP. In the current version of Autominder, the PM arbitrarily selects one of these (the first one it finds). If subsequent execution is not consistent with the STP selected, then the DTP will attempt to find an alternative consistent solution. A more principled approach would select solutions in an order that provides the greatest execution flexibility. For example, the solution that involves watching the 8:00 a.m. news leaves open the possibility of instead watching the news at 11:00 a.m. However, if the first solution found instead involved watching the later news show, then after an execution failure there would be no way to recover, as it would be too late to watch the 8:00 a.m. news. Unfortunately selecting DTP solutions to maximize flexibility is a difficult problem [26].

Epilitis combines and extends previous approaches in DTP solving, in particular by adding no-good learning. As a result, it achieves a speed-up of two orders of magnitude on a range of benchmark problems [24]. For our current Autominder scenarios, which typically involve about 30 actions, Epilitis nearly always produces solutions in less than one second, a time that is well within the bounds we require. Details of the Epilitis system can be found in [24,25].

5 Autominder

In the Autominder architecture (depicted in Figure 5), other important components rely on as well as support the PM, notably, a Client Modeler (CM) and a Personal Cognitive Orthotic (PCO). We briefly describe each of these in turn.

The Client Modeler is responsible for two tasks: (i) inferring what planned activities the client has performed, given sensor data; and (ii) learning a model of the client's expected behavior. These tasks are synergistic, in that the client model developed is used in the inference task, while the results of the inference are used to update the model.

The client's expected behavior is represented with a new formalism for temporal reasoning under uncertainty, Quantitative Temporal Dynamic Bayes Nets (QTDBNs) [4]. QTDBNs combine a simplified form of time nets with an augmented form of Dynamic Bayes nets (DBN), thereby supporting reasoning about changing conditions (fluents) in settings where there are quantitative constraints amongst them. For the purposes of the current paper, it is sufficient to consider the Dynamic Bayes net component, which represents all the client's activities for one day. The nodes in each time slice of the DBN are random variables representing (i) the incoming sensor data (e.g., client has moved to kitchen); (ii) the execution of planned activities (e.g., client has started breakfast); and (iii)

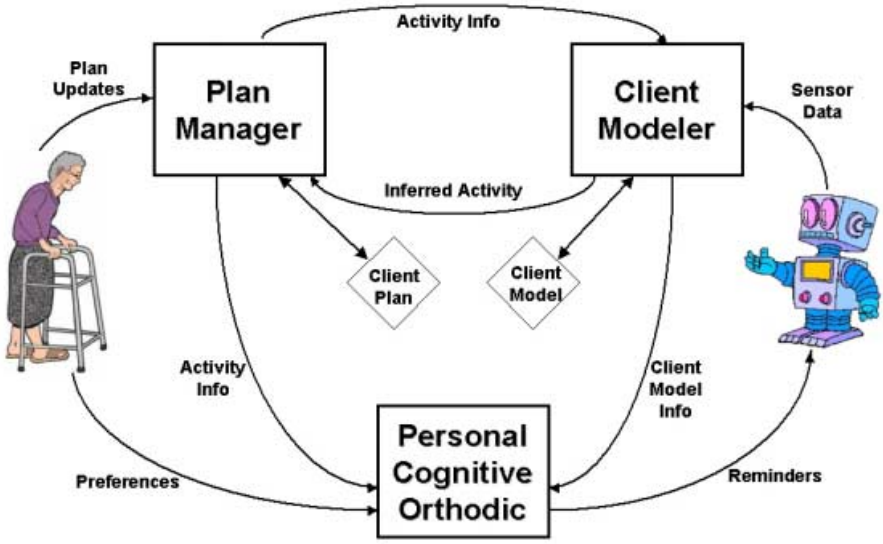


Fig. 5. The Autominder Architecture

whether a reminder for each activity has already been issued. Initially, the model is derived from the client plan by making two assumptions: first, that all activities in the plan will be executed by the client, without reminders, within the time range specified in the plan, and second, that the actual time of an activity can be described by a uniform probability density function over the range associated with that activity. The CM uses sensor data and the current time to update the model. Each time sensor data arrives, the CM performs Bayesian update. If a node representing the execution of some activity execution node's probability rises above a threshold, the activity is believed to have occurred, and the CM notifies the PM of the executed activity.

The Personalized Cognitive Orthotic (PCO) [11] uses information from the Plan Manager and Client Modeler to decide what reminders to issue and when. The PCO identifies those activities that may require reminders based on their importance and their likelihood of being executed on time as determined by the CM. It also determines the most effective times to issue each required reminder, taking account of the expected client behavior and any preferences explicitly provided by the client and the caregiver. Finally, the PCO is also designed to enable the generation of justifications for reminders. In generating a justification for a reminder, the PCO can make use of the underlying client plan, the current reminder plan, and the preferences of the caregiver and the client.

The client and caregiver communicate with Autominder via a graphical user interface (see Figure 6). On the right of the screen the client can view her daily

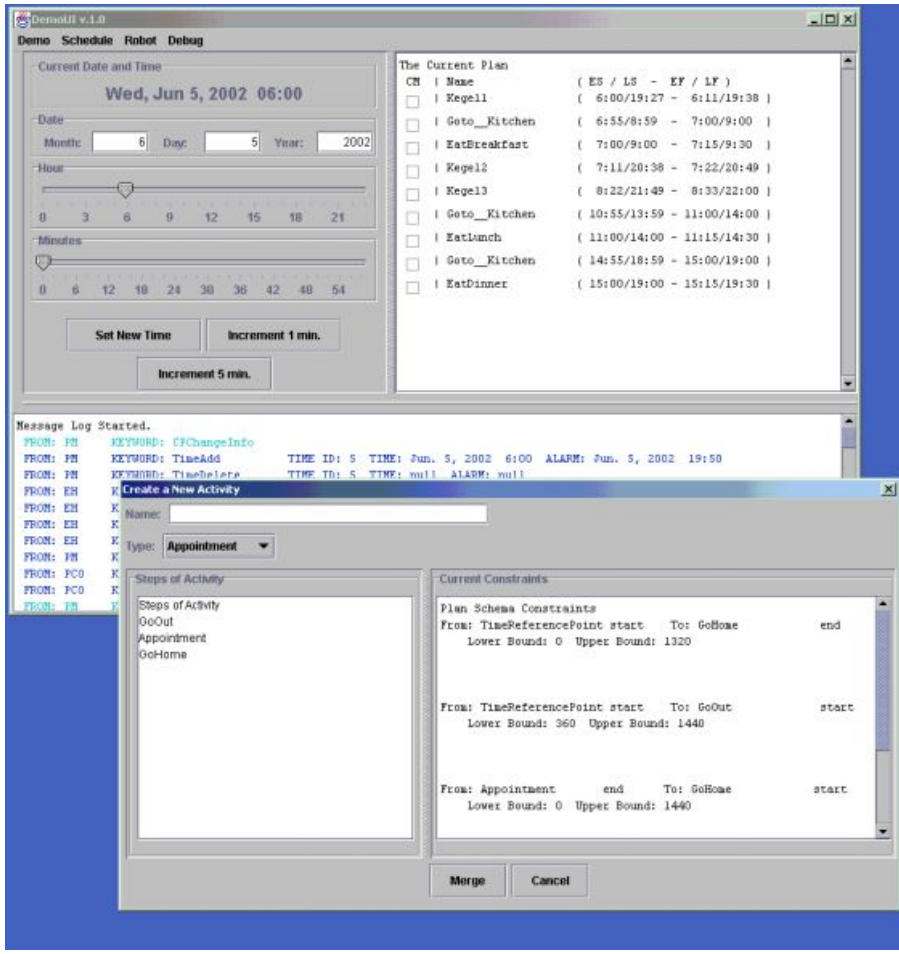


Fig. 6. Autominder: Adding new activities to the daily plan

activities, including the earliest and latest start times of each activity. Checkboxes allow the client to directly communicate to the system that he or she has completed an activity, rather than depending on sensor data to identify and confirm this information. To add new activities and constraints at run time, the client can use a drop-down menus. In this scenario, the caregiver is adding a doctor's appointment for 8 a.m. (see lower screen).

A prototype version of Autominder has been fully implemented in Java and Lisp. It includes all of the components and most of the features described above; however, the client model does not yet learn client behavior over time, and the PCO does not yet handle preferences nor issue justifications for reminders. An earlier version of the system was installed on the robot platform (Pearl),

and underwent field testing in June, 2001; we are currently in the process of integrating our newer version on Pearl.

6 Nursebot

The Autominder cognitive orthotic is being developed as part of the Initiative on Personal Robotic Assistants for the Elderly, a multi-university, multi-disciplinary research effort conceived in 1998.[3] The initial focus of the Initiative is to design an autonomous mobile robot that can "live" in the home of an older individual, and provide him or her with reminders about daily plans. To date, two prototype robots have been designed and built by members of the initiative at Carnegie Mellon. Pearl, the more recent of these robots, is depicted in Figure 7. Pearl is built on a Nomadic Technologies Scout II robot, with a custom-designed and manufactured "head", and includes a differential drive system, two on-board Pentium PCs, wireless Ethernet, SICK laser range finders, sonar sensors, microphones for speech recognition, speakers for speech synthesis, touch-sensitive graphical displays, and stereo camera systems [2,23,18]. Members of the Initiative also have interests both in other ways in which mobile robots can assist older people (e.g., telemedicine, data collection and surveillance, and physically guiding people through their environments).

Fig. 7. Pearl: A Mobile Robot Platform for the Autominder Cognitive Orthotic. Photo courtesy of Carnegie Mellon University

There are numerous reasons for embedding the Autominder system on a robotic platform in the Nursebot project. First of all, although handheld systems are popular, they are more likely to be lost or forgotten. Second, the cost of developing a robotic assistant is less expensive than retrofitting a home. At the same time, mobility is not lost since Pearl can navigate through the home.

[3] In addition to the University of Michigan, the initiative includes researchers at the University of Pittsburgh and Carnegie Mellon University.

Finally, the sensor capabilities of the robot allow for the monitoring of the client's action, thus providing integral information to the action inference engine.

7 Related Work

The large literature on workflow systems [9,10,16] is relevant to Autominder since both systems are designed to guarantee that structured tasks are performed by humans in a timely manner. The emphasis in workflow systems, however, tends to be quite different from that in Autominder. Much of the research on workflow systems focuses on the transfer of information between people in an organization who are jointly responsible for carrying out the tasks, and on the transformation of that information to enable its processing by different computer systems (interoperability). Workflow systems do not typically provide capabilities for action inference, learning of behavior patterns, or sophisticated reasoning about reminders. However, recent efforts to integrate AI planning technology with workflow tasks [7,3] may lead to results that can be integrated into Autominder.

The literature on cognitive orthotics for persons with cognitive deficits is relevant to Autominder, but not described in detail here; see [5] for an overview. Other work in plan management in the medical domain is being done by Miksch et al.[12,8]. In their work they are focusing on using planning techniques to respond to the practical demands of planning to achieve clinical guidelines. They use the Asbru representation language to check the temporal, clinical, and hierarchical consistency of plans. While our application domain that of clients with mild memory impairment, their work is geared towards assisting the caregivers.

Finally, we have already pointed to previous literature on DTPs [17,22,24]. The Autominder system also builds on prior work on the dispatch of disjunctive plans [14,27,26]. This work has primarily been focused on plans that are represented as STPs.

8 Conclusion

Execution-time plan management is essential to many systems in dynamic environments. We have described our work on plan management in the context of Autominder, a planning and reminding system designed to aid elderly persons with memory impairment. Autominder's PM builds on our earlier work on plan management, providing the tools to store and update plans representing the activities that the client is expected to perform. However, Autominder extends the earlier work by providing significantly increased representational power, along with highly efficient algorithms for handling expressive plans.

There are a number of areas of future work that we are pursuing. One of the most important is extending the PM to handle uncontrollable events[13], activities that are not under the client's direct control, such as the time at which deliveries are made. Another extension would support optional, prioritized activities in plans. We are also developing better capabilities for plan revision. In dynamic environments, plans may need to be revised in several circumstances:

(1) when a new goal to be adopted conflicts with existing plans; (2) when the environment changes in ways that invalidate the existing plans; and (3) when the client's desires and/or preferences change in ways that affect the existing plans. We are developing plan revision algorithms for plans that include expressive temporal constraints of the kind handled by our PM; our algorithms aim to make the minimal modifications required to ensure plan correctness in the face of the kinds of changes just listed.

References

1. E. Giunchiglia, A. Armando, and C. Castellini. Sat-based procedures for temporal reasoning. In *5th European Conference on Planning*, 1999.
2. G. Baltus, D. Fox, F. Gemperle, J. Goetz, T. Hirsch, D. Margaritis, M. Montermelo, J. Pineau, N. Roy, J. Schulte, and S. Thrun. Towards personal service robots for the elderly. In *Workshop on Interactive Robots and Entertainment*, 2000.
3. Pauline M. Berry and Karen L. Myers. Adaptive process management: An AI perspective. In *Proceedings of the Workshop Towards Adaptive Workflow System*, 1998. Available at `http://www.ai.sri.com/~berry/publications/cscw98-sri.html`.
4. Dirk Colbry, Barth Peintner, and Martha E. Pollack. Execution monitoring with quantitative temporal dynamic bayesian networks. In *Proceedings of the Sixth International Conference on AI Planning Systems (AIPS)*, Toulouse, France, 2002.
5. Elliot Cole. Cognitive prosthetics: An overview to a method of treatment. *NeuroRehabilitation*, 12:39–51, 1999.
6. R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49:61–95, 1991.
7. B. Drabble, T. Lydiard, and A. Tate. Workflow support in the air campaign planning process. In *Proceedings of the Workshop on Interactive and Collaborative Planning, AIPS98*, Pittsburgh, PA, 1998.
8. G. Duftschmid, S. Miksch, and W. Gall. Verification of temporal scheduling constraints in clinical praticle guidelines. In *To appear in Artificial Intelligence in Medicine, 2002*, 2002.
9. Dimitrios Georgakopoulos, Mark Hornick, and Amit Sheth. An overview of workflow management: From process modeling to workflow autonomation infastructure. *Distributed and Parallel Databases*, 3:119–153, 1995.
10. Dirk Mahling, Noel Craven, and W. Bruce Croft. From office automation to intelligent workflow systems. *IEEE Expert*, 10(3), 1995.
11. Colleen E. McCarthy and Martha E. Pollack. A plan-based personalized cognitive orthotic. In *Proceedings of the Sixth International Conference on AI Planning Systems (AIPS)*, Toulouse, France, 2002.
12. Silvia Miksch. Plan management in the medical domain. *AI Communications*, 4, 1999.
13. Paul Morris, Nicola Muscettola, and Thierry Vidal. Dynamic control of plans with temporal uncertainty. In *International Joint Conference on Artificial Intelligence-2001*, pages 494 – 502, 2001.
14. Nicola Muscettola, Paul Morris, and Ioannis Tsamardinos. Reformulating temporal plans for efficient execution. In *Proceedings of the 6th Conference on Principles of Knowledge Representation and Reasoning*, 1998.

15. Nursebot. Nursebot project: Robotic assistants for the elderly, 2000. Available at `http://www.cs.cmu.edu/~nursebot/`.
16. Gary J. Nutt. The evoluation towards flexible workflow systems. *Distributed Systems Engineering*, pages 276–294, 1996.
17. Angelo Oddi and Amedeo Cesta. Incremental forward checking for the disjunctive temporal problem. In *European Conference on Artificial Intelligence*, 2000.
18. J. Pineau and S. Thrun. High-level robot behavior control using POMDPS. In *AAAI-02 Workshop on Cognitive Robotics*, 2002.
19. Martha E. Pollack. Planning technology for intelligent cognitive orthotics. In *Proceedings of the Sixth International Conference on AI Planning Systems (AIPS)*, Toulouse, France, 2002.
20. Martha E. Pollack and John F. Horty. There's more to life than making plans: Plan management in dynamic environments. *AI Magazine*, 20(4):71–84, 1999.
21. K. Stergiou and M. Koubarakis. Backtracking algorithms for disjunctions of temporal constraints. *Artificial Intelligence*, 120:81–117, 2000.
22. Kostas Stergiou and Manolis Koubarakis. Backtracking algorithms for disjunctions of temporal constraints. In *15th National Conference on Artificial Intelligence (AAAI)*, 1998.
23. S. Thrun, J. Langford, and V. Verma. Risk senstive particle filters. *Advances in Neural Information Processing Systems*, 14, 2002.
24. Ioannis Tsamardinos. *Constraint-Based Temporal Reasoning Algorithms, with Applications to Planning*. PhD thesis, University of Pittsburgh, Pittsburgh, PA, 2001.
25. Ioannis Tsamardinos and Martha E. Pollack. Efficient solution techniques for disjunctive temporal problems. Under review. Available from the authors upon request., 2002.
26. Ioannis Tsamardinos, Martha E. Pollack, and Philip Ganchev. Flexible dispatch of disjunctive plans. In *To appear in the 6th European Conference on Planning*, 2001.
27. R. J. Wallace and E. C. Freuder. Dispatchable execution of schedules involving consumable resources. In *Proceedings of the 5th International Conference on Artificial Intelligence Planning and Scheduling*, 2000.

Path Planning for Cooperating Robots Using a GA-Fuzzy Approach

Dilip Kumar Pratihar[1] and Wolfgang Bibel[2]

[1] Regional Engineering College, Durgapur-713 209, India
dilippratihar@mailcity.com
http://www.angelfire.com/pa5/dilippratihar/mmypage
[2] Darmstadt University of Technology, FG Intellektik, FB Informatik
Alexanderstr. 10, 64283 Darmstadt, Germany
bibel@informatik.tu-darmstadt.de
http://www.inferenzsysteme.informatik.tu-darmstadt.de/~bibel/

Abstract. We consider the difficult path planning problem for a team of robots working in the same workspace. Their navigation movements are determined by the fuzzy logic controllers (FLCs) having a common knowledge base which consists of membership function distributions and fuzzy rules. Such an FLC requires the design of an appropriate knowledge base. We propose, in this paper, to automate this design task by use of a genetic algorithm (GA) which selects some good rules from a large rule base using the information of membership function distributions of the variables. Results of computer simulations are given which demonstrate the feasibility of this approach.

1 Introduction

Increasing demands push the robotics research into the direction of several autonomous and intelligent agents which cooperate in the same workspace, also termed a *multi-agent system* (MAS) [4]. The configuration of an MAS may be classified into two groups, namely the *centralized* and *decentralized* systems. In a *centralized system*, there is a master robot and the other robots working in the same workspace will have to obey the master. On the other hand, in a *decentralized system*, each robot carries out tasks cooperatively and there is no master robot. In this paper, we have concentrated on a *decentralized MAS*. In order to carry out a particular common task, the robots have to move around and do so quickly and without collisions. The problem may be described as follows: Multiple mobile robots working in the common work-space will have to find time-optimal, collision-free paths while moving from their respective starting points to the destinations. To achieve this goal, they need to plan their individual navigation paths in a (time) optimal or at least near-optimal way.

The path for each robot is constrained by the current position, the (intermediate) destination point and the movements (direction and speed) of the collaborating robots which in realistic applications are determined from sensory data. Any such data are bound to be imprecise. This suggests to use a fuzzy

M. Beetz et al. (Eds.): Plan-Based Control of Robotic Agents, LNAI 2466, pp. 193–210, 2002.

technique [19] for the movement control in the form of a fuzzy logic controller (FLC) [10]. Such an FLC consists of two parts, the membership function distributions and a rule base. The inputs and output of the FLC are expressed in terms of membership function distributions. The representation of such functions is sometimes called the *data base* of the FLC. Thus, the knowledge base (KB) of an FLC consists of a data base (DB) and a rule base (RB). The performance of an FLC depends on its knowledge base. Although FLC is becoming more and more popular to solve this type of problems, designing its knowledge base is not an easy task. A genetic algorithm (GA) [6] had been used by several investigators for fuzzy rule generation. In this connection, the work of Karr [8], Copper and Vidal [3], Liska and Melsheimer [12], Pratihar et al. [13] are worth mentioning.

In the present work, an approach for the automatic design of optimal/near-optimal fuzzy rules using a GA is proposed and its results are compared to those of a previous approach [13] based on the GA-Fuzzy combination. The navigation problem of multiple cooperating robots is taken as a bench-mark for the comparison of the approaches.

Several attempts were made by various researchers to solve coordination problems of both manipulators as well as mobile robots. Latombe [11] provides an extensive survey on the algorithmic approaches of robot motion planning. In a *decentralized system*, all possible paths for the individual robot are considered and then the possible conflicts are resolved to determine the suitable paths for all members of an MAS. Thus, in a *decentralized planning* (also known as *decoupled planning*), the selection of a priority scheme (which determines the sequence in which the planning is to be done by different agents in an MAS) plays an important role.

Kant and Zucker [7] developed an approach (known as path-velocity decomposition) of motion planning for mobile robots in the presence of moving obstacles. In their approach, planning is done in two steps. In the first step, a suitable path is planned considering the obstacles to be static and in the second step, velocity of the robot is determined in such a way that it avoids collision with the moving obstacles. Erdmann and Lozano-Perez [5] proposed the decoupled planning of multiple robots in the configuration time-space based on a fixed priority scheme. Warren [16] used an artificial potential field approach to solve the path coordination problems of multiple robots depending on a single priority scheme. But, his approach has its inherent local minima problem. Bennewitz and Burgard [1] suggested a probabilistic method for planning collision-free trajectories of multiple mobile robots. Buck et al. [2] developed a hybrid scheme of different planning methods, in the context of robot soccer, which selects the best planning method through an empirical investigation, to solve a particular problem in the best possible way. The hybrid planning scheme that selects the most suitable planning method based on a learned predictive model was found to outperform the individual planning methods. Xi et al. [17] developed an event-based method for planning and control of multiple coordinating robots. Yao and Tomizuke [18] proposed an adaptive control method for multiple coordinating robots. Moreover,

Wang and Pu [15] used cell-to-cell mapping to find time-optimal trajectories for multiple coordinating robots, where each cell corresponds to one of the states in the moving path.

More recently, Kim and Kong [9] suggested an approach in which group intelligence of multiple mobile robots was determined as a set of fuzzy rules. Their approach consists of two steps. In the first step, a mathematical reference algorithm responsible for the group behavior is realized and numerical input-output data from the reference algorithm are collected. Clustering of the input-output data is done in the second step to identify the fuzzy system structure and the parameters.

This paper consists of five sections: Section 1 describes the problem and provides a literature survey of previous work. The possible solution of the problem using FLC is proposed in Section 2. Section 3 discusses the GA-Fuzzy approaches, in detail. The results of computer simulations are presented and discussed in Section 4. Some concluding remarks are made in Section 5. Section 6 discusses the scope for future work.

2 Solution of the Problem Using Fuzzy Logic Controller

Our aim is to design suitable controller for each of the cooperating robots. Fig. 1

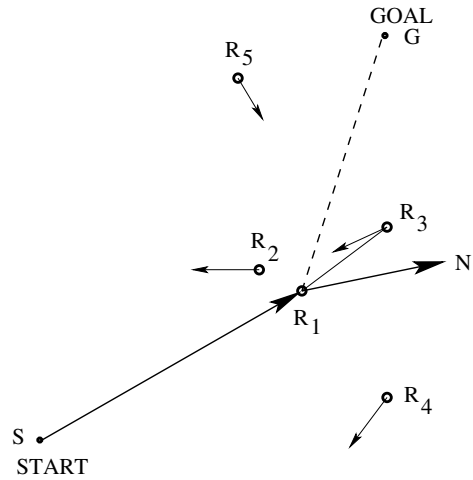


Fig. 1. A typical problem scenario consisting of five cooperating robots

shows a typical problem scenario of five cooperating mobile point-robots. Each of them is controlled by an FLC. The knowledge base of all the FLCs is kept same. It is important to mention that while planning collision-free path of a particular robot (say, R_1), the other robots (i.e., R_2,R_3,R_4,R_5) are treated as moving obstacles to it. Thus, it may be considered as a dynamic motion planning (DMP)

problem. It is to be noted that the *relative velocity* plays an important role in the DMP problems. In our approach, we do not explicitly use the relative velocity of the robot as a condition variable of the FLC to reduce the complexity of the algorithm. Instead, a practical incremental approach is adopted here to eliminate its explicit consideration. The path of a robot is considered as a collection of some small straight-line segments, each of which is traveled during a time interval. The robot makes its plan before the beginning of each time interval. A robot collects information of its neighborhood using the sensors during actual navigation. But, in computer simulation, the robot's planning is based on the predicted position of its neighbors. By knowing the present and previous positions of a robot, its predicted position is determined as $P_{pd} = P_{present} + (P_{present} - P_{previous})$. Before planning, a robot must determine its most critical neighbor. In DMP, relative velocity plays a vital role in determining the most critical neighbor of the planning robot because a neighbor physically closest to the robot may not be always critical at all. Fig. 1 shows that robot R_2 is physically nearest to the planning robot R_1, but it is not considered as a critical neighbor because its velocity directs away from the R_1's path towards the target point G. On the other hand, the direction of velocity of robot R_3 is such that it moves towards the planning robot R_1 and R_3 is considered as the most critical neighbor forward to R_1. Two condition variables, namely *distance* and *angle* are fed as inputs to the FLC and it produces one output, i.e., *deviation*. *Distance* is the distance between the robot and its most critical neighbor (i.e., R_1R_3). *Angle* is the relative angle between the path joining the robot to its target point and the path joining the robot to its most critical neighbor (i.e., angle GR_1R_3). *Deviation* is the angle through which the robot should move to avoid collision with its most critical neighbor and it is measured with respect to the line joining the robot and its target point (i.e., angle GR_1N). Condition and action variables of an FLC are expressed in terms of a membership function. Fig. 2 shows the membership function distributions of input and output variables. For simplicity, the shape of the membership function distribution is assumed to be triangular. Four grades of *distance* are chosen, namely VN (Very Near), N (Near), F (Far) and VF (Very Far). The membership function distributions for both *angle* and *deviation* are assumed to be similar and their total range is divided into five grades: L (Left), AL (Ahead Left), A (Ahead), AR (Ahead Right) and RT (Right). As there are four values of *distance* and five values of *angle*, we have considered $4 \times 5 = 20$ rules for each FLC.

Table 1 shows the author-defined rule base of the FLCs. Thus, a typical rule will look as follows:

IF *distance* is VN **AND** *angle* is L, **THEN** *deviation* is A.

2.1 Evaluation of a Solution

Each of the cooperating robots will have to plan its time-optimal path in such a way that it maintains a minimum distance (d_{min}) with other mobile robots in the work-space, to avoid collision with them. Thus, we are faced with a constrained traveling time minimization problem.

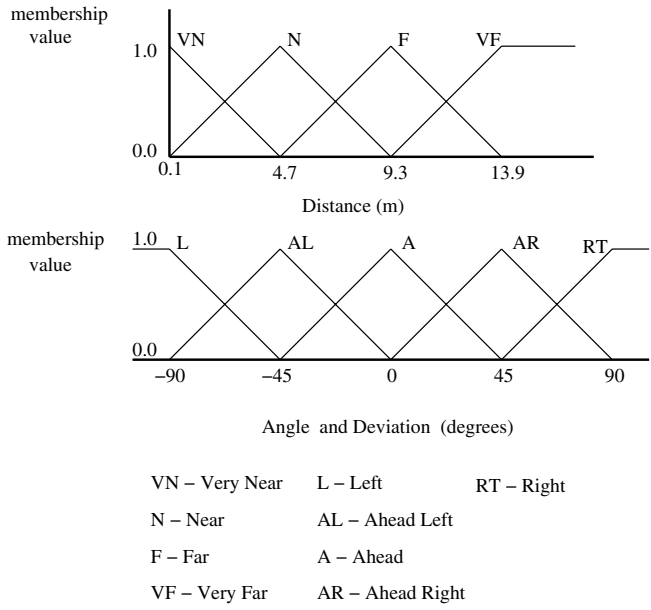


Fig. 2. Membership function distributions of input and output variables

The robot's total path is divided into a number of small straight-line segments, called *distance steps*, each traveled for a constant time interval, t. It is assumed that a robot starts from zero velocity. It accelerates during the first quarter of time interval t and then maintains a constant velocity during the next one-half of t and decelerates to zero velocity during the remaining quarter of t (refer to Fig. 3). The acceleration and deceleration rates are assumed to be equal to a. At the end of the constant velocity travel (i.e., point X in Fig. 3) in a *distance step*, the robot senses position of its neighbors and decides whether to continue moving in the same direction or to deviate from its path. If the robot has to change its path, it will decelerate to have zero velocity at the end of time interval, otherwise it will continue moving (without stopping at the end of intermediate *distance steps*) in the same direction with the velocity $V = at/4$ (the maximum velocity of the robot). It is important to mention that when the robot does not change its direction of movement in two consecutive *distance steps*, there is a saving of traveling time by $t/4$ seconds per such occasion. Continuing in this fashion, when the robot comes closer to its destination and does not find any critical neighbor ahead, it starts decelerating from a distance of $at^2/32$ (can be obtained by a trivial calculation) so as to reach its destination with zero velocity. Total traveling time T is then calculated by summing up the time intervals needed to reach its destination starting from an initial point. It is important to note that there could be several paths available for a robot to reach its destination but some of these paths may take longer traveling time. We fix a maximum traveling time T_{max}, in the algorithm. If the total traveling time

Table 1. Author-defined rule base of the FLCs

Distance	Angle	Deviation
VN	L	A
VN	AL	AR
VN	A	AL
VN	AR	AL
VN	RT	A
N	L	AL
N	AL	A
N	A	AL
N	AR	A
N	RT	AR
F	L	AL
F	AL	A
F	A	AL
F	AR	A
F	RT	AR
VF	L	A
VF	AL	A
VF	A	AL
VF	AR	A
VF	RT	A

T is found to be less than T_{max}, the objective function f during optimization is made equal to T (i.e., $f = T$). On the other hand, if the total traveling time T becomes either equal to or exceeds T_{max}, the robot is halted at its current position and its distance (d_{rem}) is determined from the destination. The objective function f is then approximately modified to $f = T_{max} + d_{rem}/V$, where V is the maximum velocity of the robot. Moreover, a minimum distance d_{min} has to be maintained between the robot and its most critical neighbor to avoid a collision between them. A penalty term, $1000/d_{min}$ is added to the objective function f to take into account this constraint and to ensure collision-free path planning during optimization. Thus, the modified objective function f' will take the form of $f' = f + 1000/d_{min}$. The same procedure is followed for all the cooperating robots and f'_i ($i = 1, 2, \ldots, R$, where R is total number of robots) is determined for all of them. Thus, the overall objective function F' can be expressed as $F' = \sum_{i=1}^{R} f'_i$.

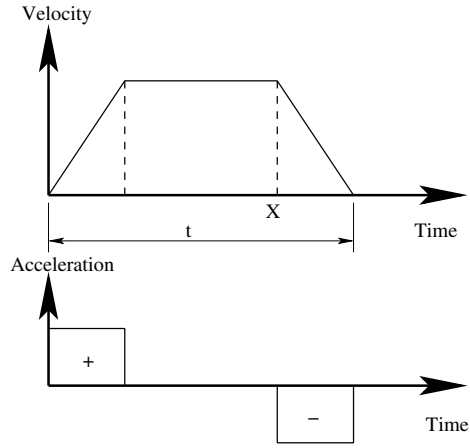


Fig. 3. Velocity and acceleration distributions

3 Proposed GA-Fuzzy Approach

In the proposed GA-Fuzzy approach, a genetic algorithm (GA) is used to improve the performance of an FLC (refer to Fig. 4). It is observed that the performance

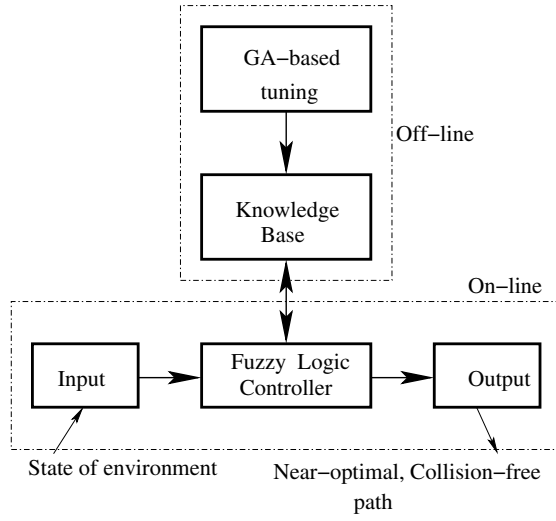


Fig. 4. A schematic showing proposed GA-Fuzzy approach

of an FLC depends mainly on its rule base, and optimizing the membership function distribution is a fine tuning process [13]. Thus, in the present work, the rule base of an FLC is optimized only using a GA and the membership function

distribution is kept unaltered. As the variables (presence or absence of a rule) are discrete in nature, a binary-coded GA [6] is used during optimization. Moreover, as a GA is computationally expensive, GA-based optimization of FLC is carried out off-line. A GA works based on the concept of fitness. In this study, fitness of a GA solution is determined as follows. We consider Q different training scenarios which are selected at random. For each of the training scenarios, the objective function F' (refer to Section 2.1) is determined and the fitness of a GA-solution is assigned as $FS = \sum_{j=1}^{Q} F'_j/Q$. GA will find through iteration a string which corresponds to the minimum fitness value as it is a traveling time minimization problem. Two different approaches based on GA-Fuzzy combinations are developed here as discussed below:

- **Approach 1: GA-based tuning of a manually-constructed FLC.** Based on our knowledge of the problem to be solved, we have designed the knowledge base of the FLCs. But, this knowledge base may not be optimal in any sense. Thus, in this approach, a GA-based tuning of the manually-constructed FLC is used [13]. The rule base of an FLC is optimized only, keeping the data base fixed. The author-defined membership function distributions and rule base of the FLCs are shown in Fig. 2 and Table 1, respectively. As there are 20 rules of an FLC, the GA-string is assumed to be 20-bits long. The presence and absence of a rule is denoted by 1 and 0, respectively. Thus, through search, the binary-coded GA will select some good rules from a manually-constructed large rule base. The optimized FLCs are obtained off-line, in this way, by a GA-based tuning. Once the optimized FLCs are available, they can be used on-line, to solve real-world problems.
- **Approach 2: Automatic design of fuzzy rules using a GA.** In approach 1, a considerable amount of time is spent to design the knowledge base of an FLC manually. Sometimes, it becomes difficult to gather knowledge beforehand of the process to be controlled using FLC. To overcome this difficulty, an approach for automatic design of fuzzy rules using a GA is proposed here. A binary-coded GA will select through search some good fuzzy rules from a large rule base consisting of the maximum number of possible rules.

  Supposing that there are two inputs (*distance* and *angle*) and one output (*deviation*) of the FLC. There are four values of *distance*, namely VN (Very Near), N (Near), F (Far), VF (Very Far) and five values (L-Left, AL-Ahead Left, A-Ahead, AR-Ahead Right, RT-Right) are assigned to both *angle* and *deviation*. The membership function distributions shown in Fig. 2 are kept unaltered in this study. The purpose of this study is to design the rule base of an FLC automatically using a GA. It is to be mentioned that no time is spent on the manual construction of the fuzzy rule base in this approach. As there are four values of *distance* and five values of *angle*, there is a maximum of $4 \times 5 = 20$ sets of condition variables. For each of these sets, there could be four possible different actions (because, it is not proper to deviate along AR if the *angle* input of FLC is AR and so on). Thus, $20 \times 4 = 80$ rules are to be dealt with during GA-based tuning. A binary-coded GA having

80-bits long string is used during tuning of fuzzy rules. It is to be noted that the presence and absence of a rule is denoted by 1 and 0, respectively. The GA-based tuning will select through iteration a set of good rules from a large rule base consisting of 80 rules.

4 Simulation Results and Discussion

The effectiveness of the proposed algorithm is tested through computer simulations. Ten ($Q = 10$) different training scenarios are selected at random and used during training. The time interval t and acceleration of the robots a are assumed to be equal to 4 *seconds* and 1 $m/second/second$, respectively. Simulation results are presented here for three different cases, as discussed below.

4.1 Case 1: Two Cooperating Robots

Two robots are moving in a grid of 36×36 m^2 in a 2-D space. Each of these robots will start from its starting point and reaches its destination without colliding each other in minimum traveling time. The minimum distance between the planning robot and its most critical neighbor (d_{min}) is kept fixed to 4 m, to avoid collision between them. Results of both approaches are presented here. The performance of a GA depends on its parameter setting. Experiments are carried out with different sets of GA-parameters and the best result is found with the following parameters, in Approach 1: crossover probability $p_c = 0.89$, mutation probability $p_m = 0.01$. We run GA for a maximum of 50 generations and the population size is set to 100. Only 4 good rules are selected by a GA from a total of 20 rules, in Approach 1. The optimized rule base of the FLCs is shown in Table 2.

It is important to mention that the data base is kept unaltered (as shown in Fig. 2) in the experiments with both the approaches. In Approach 2, p_c, p_m, population size and maximum number of generations are set equal to 0.99, 0.01, 200 and 100, respectively. These parameters are selected after a careful study. It is important to note that 9 rules are selected by a GA from a set of 80 rules. Table 3 shows the optimized rule base of the FLCs, obtained using Approach 2. The results of Approach 1 and Approach 2 are shown in Table 4 for some of the training and test scenarios. The first three rows of Table 4 show the results of three training (out of ten training scenarios) scenarios and the subsequent three rows show the results of three new test scenarios which were not considered during training. The results of a particular test scenario (no. 4 of Table 4) are shown in Fig. 5. The results of both the approaches are found to be similar, although only three rules are common in both the Tables 2 and 3. This happens because different sets of fuzzy rules may produce similar results.

4.2 Case 2: Four Cooperating Robots

Four robots are cooperating among themselves in a grid of 30×30 m^2 in a 2-D space. The initial and final points are specified for all the robots. Each

Table 2. Optimized rule base of the FLCs obtained using Approach 1 - for two cooperating robots

Distance	Angle	Deviation
N	A	AL
F	AR	A
VF	AL	A
VF	RT	A

Table 3. Optimized rule base of the FLCs obtained using Approach 2 - for two cooperating robots

Distance	Angle	Deviation
VN	A	AL
VN	AL	AR
VN	AR	A
N	A	AL
N	L	A
F	L	A
F	AR	A
VF	AL	A
VF	AR	A

Table 4. Results of two cooperating robots

Scenarios	Approach 1 Robot 1 Traveling time (seconds)	Approach 1 Robot 2 Traveling time (seconds)	Approach 2 Robot 1 Traveling time (seconds)	Approach 2 Robot 2 Traveling time (seconds)
1	37.52	35.21	37.52	35.21
2	35.98	34.62	35.98	34.62
3	36.35	34.37	36.29	34.37
4	36.96	37.13	36.96	37.13
5	29.93	30.62	29.53	30.09
6	32.97	36.55	34.15	37.55

of these robots will try to find a collision-free, time-optimal path to reach its destination. The safe and minimum distance (d_{min}) between the planning robot and its most critical neighbor is kept fixed to 1 m. In Approach 1, the following GA-parameters are selected through a detailed parametric study: $p_c = 0.94$, $p_m = 0.01$. We run GA for a maximum of 50 generations and the population size is kept fixed to 100. The optimized rule base (consisting of seven rules) obtained using Approach 1 is shown in Table 5. Thus, the GA has selected only 7 rules from a total of 20. On the other hand, GA-based tuning in Approach 2 has selected only 13 rules from a total of maximum 80 rules. We run GA with an initial population of 200 for a maximum of 100 generations and the crossover

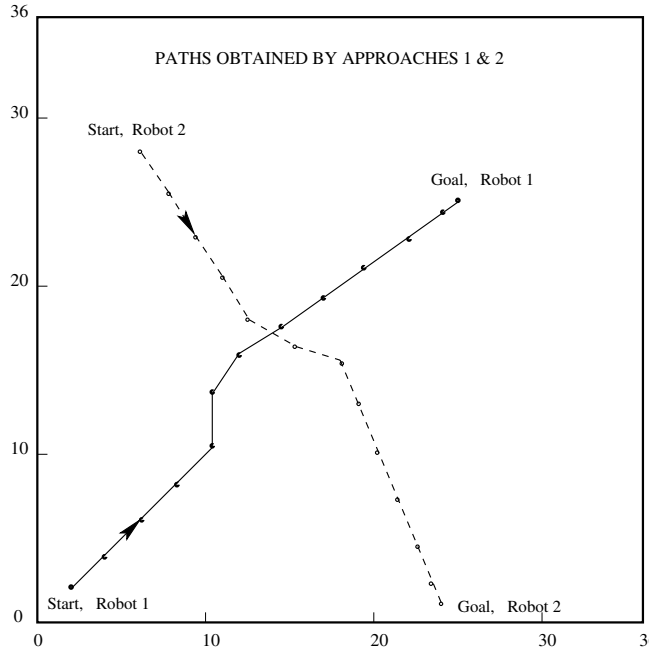


Fig. 5. Results of two cooperating robots - Approaches 1 and 2

and mutation probabilities are set equal to 0.97 and 0.02, respectively. The GA-parameters are selected through a thorough experiment with different sets of values. Table 6 shows the optimized rule base (consisting of thirteen rules) obtained using Approach 2. Four rules are found to be common in both the Tables 5 and 6. Traveling times of all four cooperating robots are shown in Table 7 for some of the training and test scenarios. The results of three training scenarios (out of ten) are shown in the first three rows of Table 7, whereas the next three rows carry traveling time information of the robots for three new test scenarios.

In most of the scenarios, Approach 2 has outperformed Approach 1, whereas in some other scenarios, Approach 1 is found to perform better than Approach 2. It is important to note that the results of Approach 1 and Approach 2 are comparable. It is also interesting to note that the GA has selected fuzzy-rules in such a way that in most of the cases, the robot will move towards left to avoid collisions with its neighbors. This particular traffic rule is evolved by a GA through iterations, although it depends on the nature of the training scenarios considered. The paths planned by all four cooperating robots for a test scenario (no. 4 of Table 7) are shown in Figs. 6 and 7 as obtained by Approaches 1 and 2, respectively.

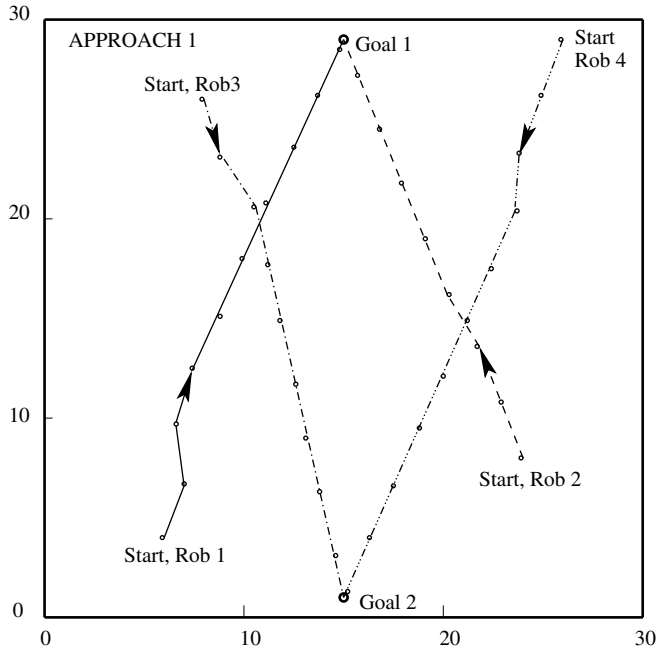


Fig. 6. Results of four cooperating robots - Approach 1

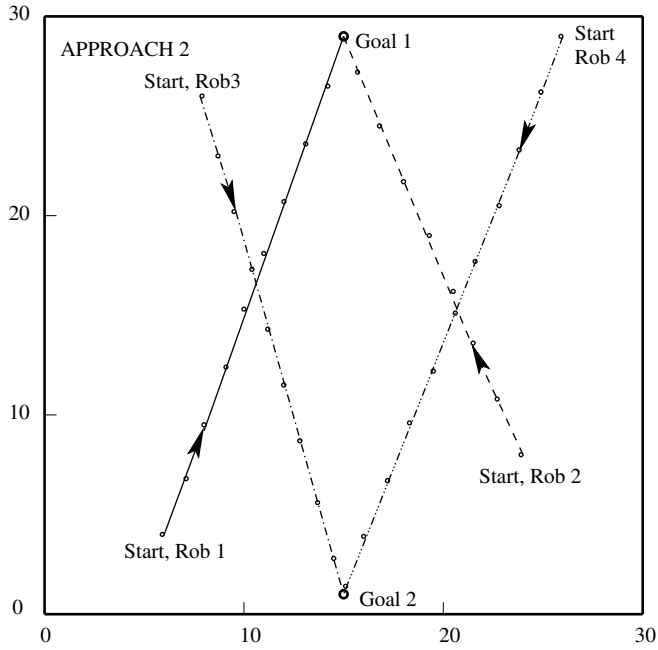


Fig. 7. Results of four cooperating robots - Approach 2

Table 5. Optimized rule base of the FLCs obtained using Approach 1- for four cooperating robots

Distance	Angle	Deviation
N	AL	A
N	AR	A
F	AL	A
F	AR	A
VF	AL	A
VF	A	AL
VF	AR	A

Table 6. Optimized rule base of the FLCs obtained using Approach 2- for four cooperating robots

Distance	Angle	Deviation
VN	AL	A
VN	L	A
VN	RT	A
N	AL	A
N	RT	A
F	AL	A
F	L	A
F	AR	A
F	RT	A
VF	AL	L
VF	L	A
VF	AR	A
VF	RT	AL

Table 7. Results of four cooperating robots

Travel	Appro 1 Robot 1 Travel time (s)	Appro 1 Robot 2 Travel time (s)	Appro 1 Robot 3 Travel time (s)	Appro 1 Robot 4 Travel time (s)	Appro 2 Robot 1 Travel time (s)	Appro 2 Robot 2 Travel time (s)	Appro 2 Robot 3 Travel time (s)	Appro 2 Robot 4 time (s)
1	28.92	26.62	27.58	26.99	26.18	24.47	27.58	29.27
2	27.23	28.85	28.95	27.43	27.23	26.62	28.95	29.59
3	30.37	23.26	31.99	29.90	30.36	21.25	29.39	27.90
4	30.06	25.46	28.96	32.38	27.43	23.46	26.61	30.11
5	28.22	24.83	29.31	32.82	25.73	22.81	27.90	30.62
6	31.89	24.21	28.04	31.67	28.14	24.21	25.73	29.39

4.3 Case 3: Twelve Cooperating Robots

Here the time-optimal and collision-free path planning problem of twelve cooperating robots is considered in a grid of 30×30 m^2. The safe distance between the planning robot and its most critical neighbor is set equal to 0.2 m, to ensure collision-free paths of the coordinating robots. In Approach 1, the following GA-parameters (which are obtained through a detailed parametric study) are found to yield the best result during optimization: $p_c = 0.96$, $p_m = 0.007$. The

population size is kept fixed to 100. We allow the GA to run for a maximum of 100 generations. During the GA-based tuning of FLCs, 9 rules are identified as good rules from a total of 20 author-defined rules. The optimized rule base is shown in Table 8. In Approach 2, the best result is found with the following

Table 8. Optimized rule base of the FLCs obtained using Approach 1 - for twelve cooperating robots

Distance	Angle	Deviation
VN	RT	A
N	AL	A
N	AR	A
F	AL	A
F	A	AL
F	AR	A
VF	L	A
VF	AL	A
VF	A	AL

Table 9. Optimized rule base of the FLCs obtained using Approach 2 - for twelve cooperating robots

Distance	Angle	Deviation
VN	AL	A
VN	L	A
N	AL	A
N	L	A
N	AR	A
N	RT	A
F	A	AL
F	AL	AR
F	L	AL
F	AR	A
VF	A	AR
VF	AL	L
VF	L	AL

GA-parameters during the GA-based tuning of FLCs: $p_c = 0.85$, $p_m = 0.009$. We run the GA (with a population size of 200) for a maximum of 100 generations. The GA has picked up 13 good rules (refer to Table 9) from a total of 80 possible rules. Only four rules are found to be common in both the Tables 8 and 9. Results of twelve cooperating robots for some of the training and test scenarios obtained using Approach 1 and Approach 2 are shown in Table 10 and Table 11, respectively. The first three rows of both the tables carry information of the three training scenarios (out of ten training scenarios) and the subsequent three rows give information of the three test scenarios. The results of both the approaches are found to be comparable by comparing Table 10 with Table 11. Fig. 8 shows the paths planned by twelve cooperating robots for a particular test scenario (no. 4 of Table 10) using Approach 1, whereas the results of the same test scenario (refer to no. 4 of Table 11) as obtained by Approach 2 are shown in Fig. 9.

It is important to mention that the task of manual construction of the fuzzy rule base in Approach 1 is replaced by a GA-based tuning of the fuzzy rule base

Table 10. Results of twelve cooperating robots – using Approach 1

	Rob1 time (s)	Rob2 time (s)	Rob3 time (s)	Rob4 time (s)	Rob5 time (s)	Rob6 time (s)	Rob7 time (s)	Rob8 time (s)	Rob9 time (s)	Rob10 time (s)	Rob11 time (s)	Rob12 time (s)
1	33.35	21.03	22.24	19.65	18.04	23.43	28.54	27.80	35.77	26.25	23.93	20.67
2	33.86	22.36	22.24	21.13	16.37	19.16	33.98	29.76	30.21	24.61	22.45	21.30
3	28.18	20.80	22.96	21.45	19.33	21.13	27.94	33.52	33.54	23.43	25.36	19.16
4	32.64	22.33	23.46	20.29	18.04	22.27	32.17	33.66	34.64	24.09	24.11	21.30
5	28.06	23.21	22.81	21.13	19.33	22.81	31.54	32.63	33.74	22.27	25.57	19.91
6	29.85	21.03	24.21	18.85	17.70	20.83	24.38	29.52	27.43	20.29	23.93	19.90

Table 11. Results of twelve cooperating robots – using Approach 2

	Rob1 time (s)	Rob2 time (s)	Rob3 time (s)	Rob4 time (s)	Rob5 time (s)	Rob6 time (s)	Rob7 time (s)	Rob8 time (s)	Rob9 time (s)	Rob10 time (s)	Rob11 time (s)	Rob12 time (s)
1	31.49	21.03	22.24	19.65	18.04	23.43	28.38	29.92	33.55	26.25	23.93	20.67
2	32.77	22.36	22.24	21.13	16.37	19.16	33.93	29.76	30.21	24.61	22.45	21.64
3	27.74	20.80	22.96	21.45	19.33	21.33	27.94	33.52	33.11	23.43	25.36	21.63
4	32.83	22.33	23.46	20.29	18.04	22.27	33.37	33.51	31.11	24.09	24.11	22.45
5	27.77	22.12	22.81	21.13	19.33	22.81	33.37	32.61	32.25	22.27	25.57	21.18
6	29.44	23.20	24.21	18.85	17.70	20.83	24.38	29.52	28.99	20.29	23.93	19.85

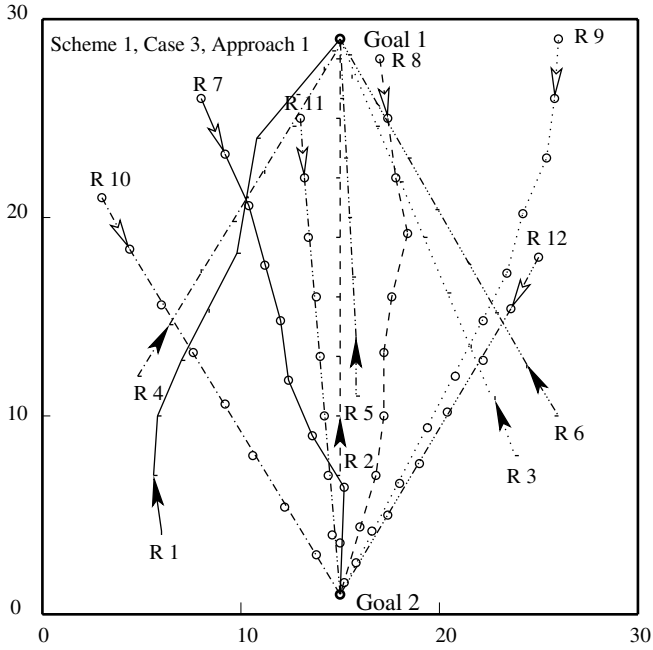


Fig. 8. Results of twelve cooperating robots – Approach 1

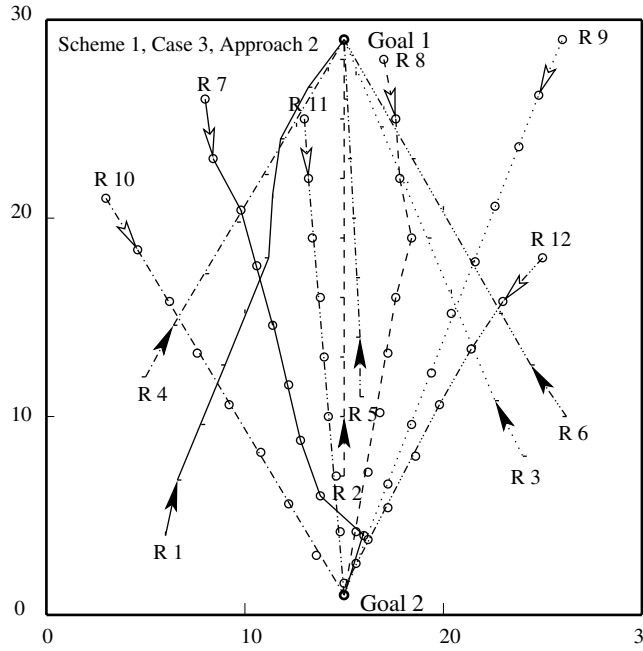


Fig. 9. Results of twelve cooperating robots – Approach 2

in Approach 2. Thus, the GA takes the responsibility of designing a reasonably good fuzzy rule base in Approach 2. In the second approach, the user need not have a thorough knowledge of the process to be controlled. This makes the second approach more interesting compared to the first. In Approach 2, the GA takes the whole responsibility of designing a good fuzzy rule base. Thus, the GA faces a tougher job in Approach 2 compared to Approach 1. The population size of GA is kept to a higher value in Approach 2 compared to that in Approach 1.

In this paper, the GA-Fuzzy approach is used to find on-line solution of the path planning problems of multiple mobile robots working in the same workspace. Using the GA-tuned FLCs, we are interested to obtain acceptable solutions only which may not be true optimum always. That is why, global search techniques like depth-first search, A^* algorithm and others have not been used, in this paper, to find the true optimum solutions. It might be interesting to obtain the true optimum solution of the path planning problem of a mobile robot in the presence of some static obstacles [14].

5 Concluding Remarks

Time optimal/near-optimal path planning problem of multiple cooperating robots is solved, in the present work, using GA-Fuzzy approaches. A GA-Fuzzy approach in which good fuzzy rules are generated automatically using a GA has

been proposed here and its performance is compared to an already proposed GA-Fuzzy approach. The results of the two approaches are found to be comparable, although Approach 2 (automatic design of fuzzy rules using a GA) might be more interesting as no time is spent here on manual construction of fuzzy rule base. The GA-based tuning is done off-line. Once the optimized FLCs are designed, off-line, using a GA, those are suitable for on-line implementations. This technique might be useful in designing a controller for each agent of a multi-agent system.

6 Scope for Future Work

The scope for future work (on which the first author is working, at present) may be summarized as follows:

- The present work is based on a single priority scheme. The priority scheme indicates the sequence in which the planning is to be done by different robots in a multi-agent system. Thus, this work may be extended to incorporate an optimal priority scheme.
- All the robots here are assumed to have the same velocity profile. But, in practice, they may have different velocity profiles. The velocities of the robots can be determined using fuzzy rules also.
- The performance of the proposed GA-based tuning of FLC has not been compared to that of the other learning methods like neural network-based tuning of FLC and others. This work may be extended for the purpose of comparison with other learning methods. Moreover, the performance of the proposed GA-Fuzzy approach may be compared to that of an algorithmic approach like potential field method, to solve the same problem.
- From the control point of view, the number of rules in the rule base of an FLC should be as minimal as possible but care must be taken so that there is no chance of weak-firing or non-firing of the rules. The technique adopted for minimization in propositional logic may also be used for minimizing the number of rules to be present in the optimized rule base of an FLC.
- For simplicity, the shape of membership function distributions for the input and output variables of an FLC is assumed to be triangular. It is understood that a smoother transition of the different control regimes can be obtained with the Gaussian distributions but at the cost of a little more computation. Thus, the triangular distributions (considered in this paper) may be replaced by the Gaussian distributions.

Acknowledgment

The first author gratefully acknowledges the financial support of the Alexander von Humboldt Foundation, Bonn, Germany, to carry out this research work.

References

1. Bennewitz, M. and Burgard, W.: A probabilistic method for planning collision-free trajectories of multiple mobile robots. Proc. of the Workshop Service Robotics – Applications and Safety Issues in an Emerging Market at the 14th ECAI (2000)
2. Buck, S., Weber, U., Beetz, M., and Schmitt, T.: Multi Robot Path Planning for Dynamic Environments: A Case Study. Proc. of the IEEE/RSJIROS Conference, Wailea, USA (2001)
3. Cooper, M.G. and Vidal, J.J.: Genetic design of fuzzy controllers: the cart and jointed pole problem. Proc. of the Third IEEE International Conference on Fuzzy Systems (1994) 1332-1337
4. Dutech, A., Buffet, O., and Charpillet, F.: Multi-Agent Systems by Incremental Gradient Reinforcement Learning. Proc. of IJCAI'01 Conference, Seattle, USA (2001) 833-838
5. Erdmann, M., and Lozano-Perez, T.: On Multiple Moving Objects. Algorithmica. **2** (1987) 477-521
6. Goldberg, D.E.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, Reading, MA, USA (1989)
7. Kant, K., and Zucker S.W.: Towards Efficient Trajectory Planning: The Path-Velocity Decomposition. The International Journal of Robotics Research. 5(3) (1986) 72-89
8. Karr, C.: Design of an adaptive fuzzy logic controller using a genetic algorithm. Proc. of the Fourth International Conference on Genetic Algorithms, Morgan Kaufmann, San Mateo, CA, USA (1991) 450-457
9. Kim, J.H. and Kong, S.G.: Fuzzy Systems for Group Intelligence of Autonomous Mobile Robots. Proc. of IEEE Intl. Fuzzy Systems Conference, Seoul, Korea (1999) I-100-I-104
10. Kosko, B.: Neural Networks and Fuzzy Systems. Prentice-Hall, New Delhi, India (1994)
11. Latombe, J.C.: Robot Motion Planning. Kluwer Academic Publishers, Norwell, MA (1991)
12. Liska, J., and Melsheimer, S.S.: Complete design of fuzzy logic systems using genetic algorithms. Proc. of the Third IEEE International Conference on Fuzzy Systems (1994) 1377-1382
13. Pratihar, D.K., Deb, K., and Ghosh, A.: A Genetic-Fuzzy Approach for Mobile Robot Navigation Among Moving Obstacles. Intl. Journal of Approximate Reasoning. **20** (1999) 145-172
14. Pratihar, D.K., Deb, K., and Ghosh, A.: Fuzzy-Genetic Algorithms and Time-Optimal Obstacle-Free Path Generation for Mobile Robots. Engineering Optimization. **32** (1999) 117-142
15. Wang, F.Y., and Pu, B.: Planning Time-Optimal Trajectory for Coordinated Robot Arms. Proc. of IEEE Intl. Conf. on Robotics and Automation. **1** (1993) 245-250
16. Warren, C.: Multiple Robot Path Coordination Using Artificial Potential Fields. Proc. of the IEEE International Conference on Robotics and Automation (ICRA) (1990) 500-505
17. Xi, N., Tarn, T.J., and Bejczy, A.K.: Event-Based Planning and Control for Multi-Robot Coordination. Proc. of IEEE Intl. Conf. on Robotics and Automation. **1** (1993) 251-258
18. Yao, B., and Tomizuka, M.: Adaptive Coordinated Control of Multiple Manipulators Handling a Constrained Object. Proc. of IEEE Intl. Conf. on Robotics and Automation. **1** (1993) 624-629
19. Zadeh, L.A.: Fuzzy Sets. Information and Control. **8** (1965) 338-353

Performance of a Distributed Robotic System Using Shared Communication Channels*

Paul Rybski, Sascha Stoeter, Maria Gini,
Dean Hougen, and Nikolaos Papanikolopoulos

Department of Computer Science and Engineering
University of Minnesota
{rybski,stoeter,gini,hougen,npapas}@cs.umn.edu

Abstract. We have designed and built a set of miniature robots and developed a distributed software system to control them. We present experimental results on a surveillance task in which multiple robots patrol an area and watch for motion. We discuss how the limited communication bandwidth affects robot performance in accomplishing the task and analyze how performance depends on the number of robots that share the bandwidth.

1 Introduction

Controlling a group of miniature mobile robots in a coordinated fashion can be a very challenging task. The small size of the robots greatly limits the power of on-board computers they can carry and the amount of sensor processing they can do. One way to overcome these limitations is to use a robust communications link between the robots and a more powerful off-board processor, where more extensive computation can be done. Unfortunately, the robots' small size also limits the bandwidth of the communications system that they can employ.

We describe a case study of a group of bandwidth-limited miniature robots, called Scouts [16]. A Scout, shown in Figure 1, is a cylindrical robot 11.5 cm in length and 4 cm in diameter. Scouts can transmit video from a small camera to a remote source for processing. They can transmit and receive digital commands over a separate communications link that uses an ad-hoc packetized communications protocol. Scouts move in two ways. They can use their wheels to travel over smooth surfaces (even climbing a 20 deg slope) and are capable of jumping over objects 30 cm in height using their spring-loaded tails.

Scouts are designed for surveillance and reconnaissance tasks where they can be remotely teleoperated by a human controller. The analog video provides a real-time update of the Scout's surroundings to the teleoperator. The small size of the Scouts gives them the ability to access low overhangs and small passages and send back useful data. However, this small size is a disadvantage because

* Material based upon work supported in part by the DARPA/MTO Office, ARPA Order No. G155, Program Code No. 8H20, issued by DARPA/CMD under Contract #MDA972-98-C-0008. Parts of this paper were presented at SPIE 2001, Sensor Fusion and Decentralized Control in Robotic Systems IV, October 2001.

M. Beetz et al. (Eds.): Plan-Based Control of Robotic Agents, LNAI 2466, pp. 211–225, 2002.

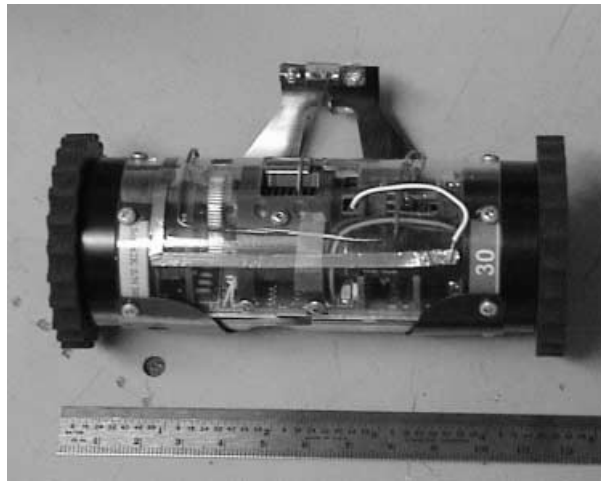


Fig. 1. The Scout robot shown next to a ruler (in cm) for scale

it limits the Scout's on-board computational and communications capabilities. This problem is exacerbated by the amount of volume that the robot's motors, actuators, and power supply take up. Additionally, the Scout's power budget must be split between the actuators, the processor, the camera, and the communications systems. Thus, the processor that the Scouts employ is specifically set to run at a low clock speed to conserve as much power as possible.

Since Scouts do not have the space or power for the electronics necessary to process their own video signals, all of their sensory data must be broadcast to a human operator or to a remote computer (possibly on a larger robot) for processing. Actuator commands are then relayed back to the miniature robots. The operation of the Scouts is completely dependent on the RF communications links they employ. In order to handle high demand for this low capacity communications system, we have developed a novel process management/scheduling system.

In order for Scouts to process their own video signals, they would most likely need to make use of a small digital video system, such as is found in many commercial USB cameras. However, this would restrict the use of the Scout to the relatively simple application described in this paper. The Scout is very useful as a teleoperated device where a human receives the transmitted video and pilots the Scout from a remote location. The analog video transmitter is vital in this case because of the ability to receive video data in real time. If the Scout were to transmit digital information it received from a digital camera, the time between successive frames of video would be greatly reduced.

Scouts receive command packets transmitted by a radio hooked up to a remote computer. Each Scout has a unique network ID, allowing a single radio frequency to carry commands for multiple robots. By interleaving packets destined for the different robots, multiple Scouts can be controlled simultaneously.

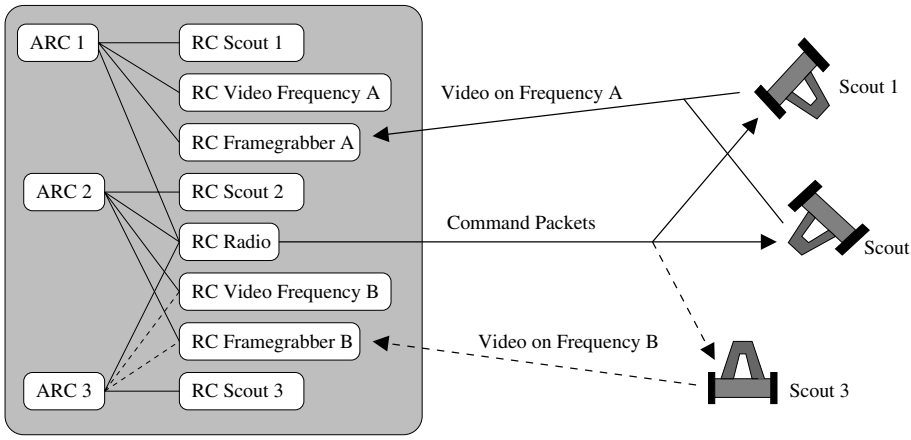


Fig. 2. Three Scouts are controlled by several ARCs. Scout 1 uses video frequency A while Scouts 2 and 3 share video frequency B and thus cannot be used simultaneously. Solid lines indicate active connections between components while dashed lines indicate connections that are not currently active but may be active later

Our communications network operates on a single carrier frequency, with a command throughput of 20-30 packets/second.

Video data is broadcast over a fixed-frequency analog radio link and must be captured by a video receiver and fed into a framegrabber for digitizing. Because the video is a continuous analog stream, only one robot can broadcast on a given frequency at a time. Signals from multiple robots transmitting on the same frequency disrupt each other and become useless. Only two different video frequencies are available with the current Scout hardware. As a result, video from more than 2 robots can be captured only by interleaving the time each robot's transmitter is on. Thus, an automated scheduling system is required to ensure that the robots share the communications resources.

2 Dynamic Allocation of Resources

The decision processes that control the Scouts need to connect to the individual resources that are necessary to control the physical hardware. We have designed a distributed software architecture [19], which dynamically coordinates hardware resources across a network of computers and shares them between client processes.

In order for a process to control a single Scout, several physical resources are required. First, a robot which is not currently in use by another process must be selected. Next, a command radio is needed which has the capacity to handle the requested throughput from (possibly) multiple different control processes. If the Scout is to transmit video, exclusive access to a fixed video frequency is required together with a framegrabber connected to a tuned video receiver. Each

instance of these four resources is managed by its own Resource Controller (RC). Resources that can only be managed by having simultaneous access to groups of RCs are handled by a second layer of components called Aggregate Resource Controllers (ARCs). An ARC is only allowed to control its Scout if it has access to four RCs of the appropriate type.

Figure 2 illustrates the interconnections between the components in the system. In this example, three Scouts are being controlled by three different ARCs. Scout 1 has full access to frequency A while Scouts 2 and 3 share video frequency B. Video receivers tuned to the two frequencies are attached to framegrabber RCs. Because Scout 1 does not have to share its video frequency with any other Scout, it can operate continuously without interference. Scouts 2 and 3 must share the video frequency bandwidth and so the time that they can access their respective RCs is scheduled so they do not interfere with one another. In the figure, ARC 2 is currently in control of Scout 2 while ARC 3 must wait for its turn to run.

The system tries to grant simultaneous access to as many ARCs as possible. ARCs are divided into sets depending on the RCs they request. ARCs that ask for independent sets of RCs are put into independent groups which are allowed to run in parallel. The ARCs that have some RCs in common are examined to determine which ones can operate in parallel and which are mutually exclusive. ARCs which request the same sharable RC may be able to run simultaneously, as long as the capacity requests for the sharable RC do not exceed its total capacity. ARCs which request a non-sharable RC must break their operating time into slices.

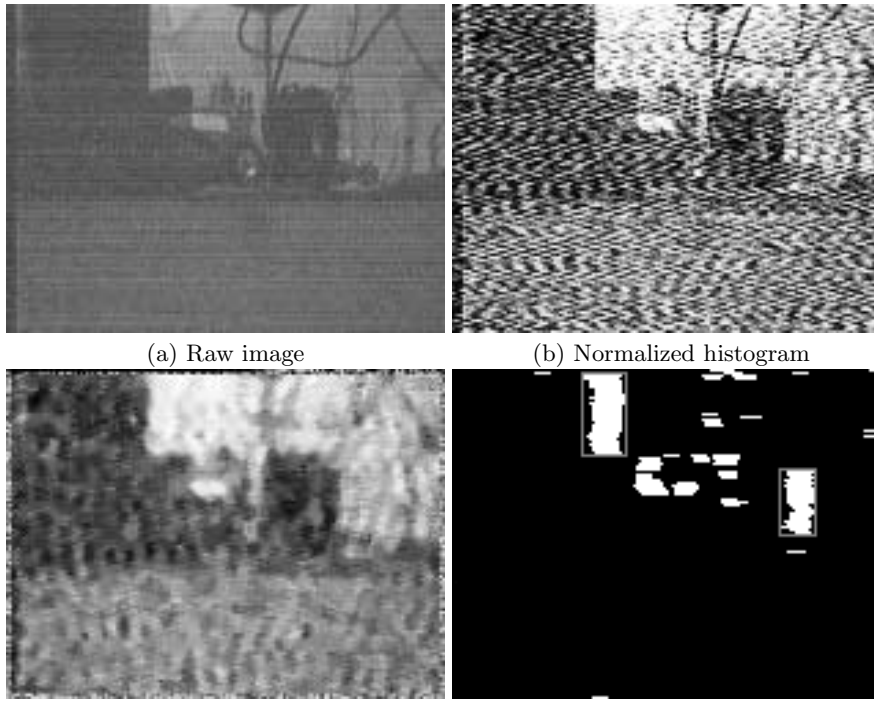
Sharable RCs, such as the Scout radio, manage their own schedules to ensure that each of the ARCs requesting them is given access at the requested rate. When requesting access to a sharable RC, an ARC must specify how often it will make requests. To simplify the scheduling process, sharable RCs can be requested only with a constant interval between invocation, and each request must complete before the next request is made.

3 A Distributed Surveillance Task

In the experiments we describe the Scouts are deployed into an area and watch for motion. This is useful in situations where it is impractical to place fixed cameras because of difficulties relating to power, portability, or even the safety of the operator.

Before watching for motion, Scouts must position themselves in a safe area, such as under a chair or a table, or close to a wall. This is needed because the miniature size of the Scouts will make them easy targets for removal or easily crushed if they were to stay in an open area.

To locate a dark area a Scout spins in a circle using discrete movements and collects images. The mean value of the pixels in each image is computed and then used to determine the location of the darkest area of the room. Once a dark area has been identified, the Scout moves toward it by analyzing a strip in the image along the horizon and determining the horizontal position of the darkest area



(a) Raw image (b) Normalized histogram

(c) Median filtered (d) Difference of two frames

Fig. 3. Calculating motion from noisy video. The raw image (a) is first normalized (b) in order to accentuate contrast. Unfortunately, this accentuates noise from RF interference in the video signal, so the image is median filtered (c) before being compared against another image to determine whether the robot has moved (d)

within that strip. The Scout stops when its camera is pressed against a dark object or is in a shadow. If none of these conditions arises, the Scout will quit moving after a minute or two of operation.

Detecting moving objects is accomplished using frame differencing. The Scout stays still and subtracts sequential images in the video stream to determine whether the scene has changed at all. Pixels that are changed are grouped together into contiguous regions. RF noise can cause a great deal of perceived motion between frames. This is filtered out by analyzing the shapes and sizes of the regions and ignoring regions that are caused by noise. Currently, we use a hand-tuned filter.

Motion detection is used also for collision detection. If the Scout drives into an obstacle, all motion in the image frame will stop. If no motion is detected after the Scout attempts to move, it will start moving in random directions in an attempt to free itself. In addition to freeing the Scout, this random motion

has the additional benefit of changing the orientation of the antenna. This often improves reception and so helps obtain better quality images.

For image processing, we compute the grayscale histogram of the Scout video and normalize it in order to accentuate the contrasts between light and dark areas (Figure 3(b)). We then smooth the image by using a 5×5 median filter (Figure 3(c)). Figure 3(d) shows the result of performing frame differencing and connected region extraction between two consecutive images. The white regions are connected regions that are different between the two images. The two gray rectangles highlight regions that are considered to be caused by the Scout's motion. All of the other regions could have been caused by random RF noise (this is decided by analyzing their shape and area) and are thus ignored.

4 Experimental Results

We examined the Scouts' ability to accomplish the surveillance task with a series of experimental runs. These experiments were designed to test the performance of the Scouts and the controlling architecture in a number of situations and group configurations. In particular, we were interested in evaluating the effectiveness of the vision-based behaviors for navigating the Scouts to useful hiding positions, the effectiveness of the motion detection algorithm under different conditions, the efficacy of the scheduling system in scheduling access to the communications bandwidth, and the overall performance of the robotic team in detecting motion.

4.1 Hiding and Viewing a Room

To test the ability of the Scouts to operate in a real-world environment, we set up a test course in our lab using chairs, cabinets, boxes, and miscellaneous other materials. The goal of each Scout in these experiments was to find a suitable dark hiding place, move there, and turn around to face a lighted area of the room.

The environment shown in Figure 4 is 6.1 m by 4.2 m wide and has a number of secluded areas in which the Scout could hide. The Scouts were started at the center of one of the 16 tiles in the center of the room and were pointed at one of 8 possible orientations. Both the position index and orientation were chosen from a uniform random distribution. The *Hiding and Viewing experiment* was divided into three cases, each using a different number of Scouts or communication channels:

(1) a single Scout on a single video frequency. This case was selected to serve as a baseline.
(2) two Scouts that had to share a single video frequency using the scheduler.
(3) two Scouts, each on its own video frequency.

Within each case, ten trials were run. The stopping positions and orientations of the Scouts at the end of the trials were used later for the *Detect Motion experiment*. They are shown later in Figure 5.

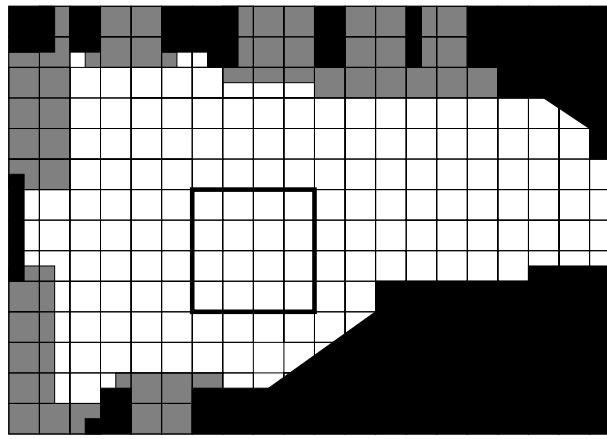


Fig. 4. A grid layout of the 6.1 m by 4.2 m room where the experiments were conducted. The white squares indicate open space, the gray areas indicate areas where Scouts could hide (underneath tables and benches), and the black areas indicate obstacles/obstructions. The black square outline in the center of the open area shows the area where the Scouts were started

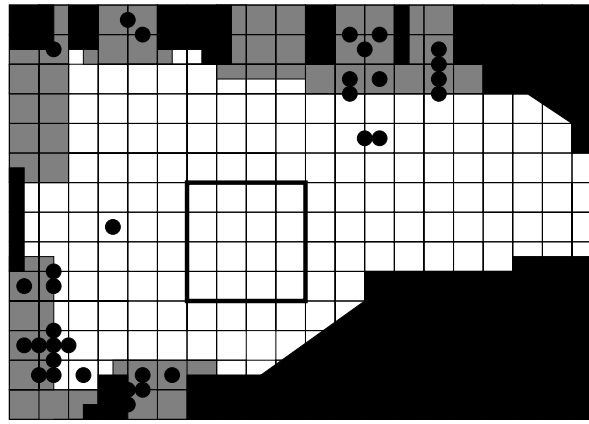


Fig. 5. Hiding and Viewing experiment. Positions that the Scouts found for themselves in the 6.1 m by 4.2 m room are represented as dots

The amount of time each behavior could use the video frequency was specified as ten seconds. When a behavior was granted access to the video frequency and could control its Scout, it had to wait three seconds before actually being able to make use of the video signal. This was done because the Scout camera requires 2-3 seconds of warm-up time before the image stabilizes. Thus, Scouts effectively had seven seconds of useful work time when they were granted access to all of their resources.

Figure 5 shows the hiding places found for all trials of all cases. Over all the trials, the Scouts were able to hide themselves 90% of the time. In the remaining 10% of the time, the Scouts reached a 60 second time-out, and stopped out in the open where they could be more easily seen and bumped into. This time-out was required because the Scouts are unable to determine with confidence if progress is being made in moving towards a hiding position. This time-out was also encountered on some successful hiding trials, as the Scout continued to try to progress to a darker hiding position, even after reaching cover. For this reason, the non-hiding trials are not considered outliers in the time data.

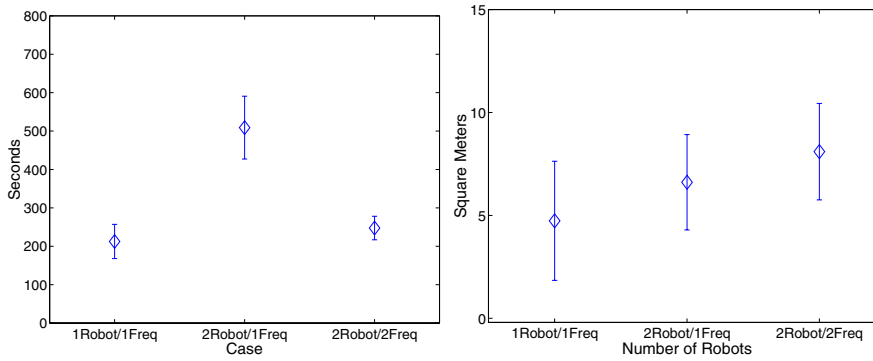


Fig. 6. Hiding and Viewing experiment. On the left, the average time (in seconds) to complete each trial. On the right, the total area (in square meters) in the field of view of the Scouts. Three cases are shown: one Scout, two Scouts on a single frequency and two Scouts on two different frequencies. The plots show the means and standard deviations of the data from the different cases

Once the Scouts had positioned themselves in the environment, they attempted to orient themselves to view a lighted area of the room. Figure 6 shows the times for the Scouts to reach their final poses (positions and orientations) in each trial of each case of the experiment. In the cases with two Scouts per trial, each value plotted is the average time the Scouts took to reach their final poses. As can be seen from this figure, two Scouts on a single video frequency (second case) took longer to reach their final poses than did a single Scout (first case). This is to be expected—the Scouts are time-multiplexing the video frequency resource. There is also a somewhat greater average time for two Scouts on two different video frequencies (third case) to reach their final poses than there is for the single scout case (for case 1, mean = 212.50, $\sigma = 44.55$; for case 3, mean = 247.50, $\sigma = 30.62$), however, these differences are not statistically significant at the 95% confidence level (two-tailed, two-sample t test, $p = 0.0555$).

One interpretation of these results is that one Scout is better than two on the same frequency (as the task is accomplished more quickly by one) and that one Scout and two on different frequencies are approximately equal on this task.

However, this ignores the fact that using two Scouts increases robustness and that two Scouts would likely be able to accomplish more than a single Scout.

Nonetheless, even if two Scouts could accomplish twice as much as one after reaching their final poses, one Scout is still better, on average, than two on the same frequency, as the time for the second case is significantly greater than twice the time for the first case. This is because when sharing the bandwidth for video transmission, up to 30% of the time is lost waiting for the video transmitter to warm up. For this reason, deploying two Scouts sequentially would often make more sense than deploying them in parallel, if the Scouts must share the video frequency.

In contrast, if two Scouts could accomplish twice as much as one after reaching their final poses, then two Scouts on different frequencies would be better than one, because of the additional reliability they provide, as we will see later in the Detect Motion experiment. However, the assumption that two Scouts are twice as good as one is unlikely to hold true.

Since the overall mission is surveillance, one measure of Scout performance after deployment is the area in their field of view. Figure 6 shows the total area viewed by the Scouts for each case. Considering the area viewed, two Scouts on different frequencies are again better than one for this task, as the area viewed is larger in case 3 than in case 1 (for case 1, mean = 4.73, $\sigma = 2.89$; for case 3, mean = 8.09, $\sigma = 2.34$)—this difference is significant (one-tailed, two-sample t test, $p = 0.0053$).

4.2 Detecting Motion

A second experiment was run to test the Scouts' abilities to detect motion. Four different cases were tested:

(1) a single Scout using a single video frequency,
(2) two Scouts sharing a single video frequency,
(3) two Scouts using two different video frequencies, and
(4) four Scouts sharing two different video frequencies.

For each of the four cases, the Scouts were placed in ten different positions. These positions were the same as the hiding positions obtained in the previous experiment. In the case using four Scouts, for which no hiding experiment was run, the positions were randomly sampled with replacement from the results of the other hiding experiments. In each position, five individual motion detection trials were run, bringing the total number of individual trials to two hundred.

The moving target the Scouts had to detect was a Pioneer 1 mobile robot. A Pioneer was chosen for its ability to repeatedly travel over a path at a constant speed. This reduced some of the variability between experiments that the use of human subjects might have caused. Additional experimentation showed that the Scouts were at least as good at detecting human motion as they were at detecting the Pioneer. The Pioneer entered the room from the right and made its way over to the left. The Pioneer moved at a speed of approximately 570 mm/s and traversed the length of the room in 8.5 seconds on average. Once it had

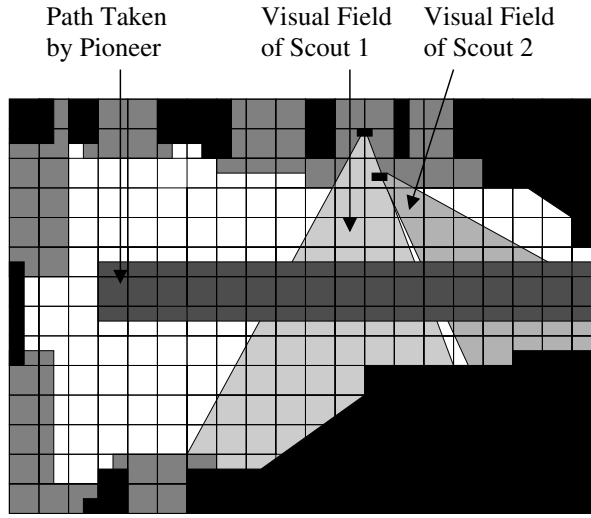


Fig. 7. Detect Motion experiment. Example Scout placement in the room. In this instance, two Scouts view the path of the Pioneer robot, shown in dark gray in the middle of the room. The fields of view do not happen to overlap between these two Scouts

moved 4.9 m into the room, it turned around and moved back out again. With a 4 second average turn time, the average time the Pioneer was moving in the room was 21 seconds.

Figure 7 illustrates the fields of view seen by two Scouts and the area of the Pioneer's path that they cover. While the views of these Scouts do not overlap, there was a large amount of overlap in some of the other placements of Scouts.

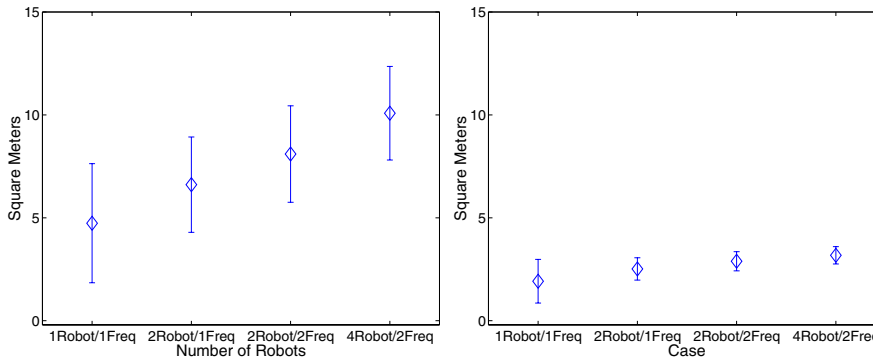


Fig. 8. Detect Motion experiment. On the left is the area of the field of view of the Scouts in each case. On the right is the area of target motion within the field of view

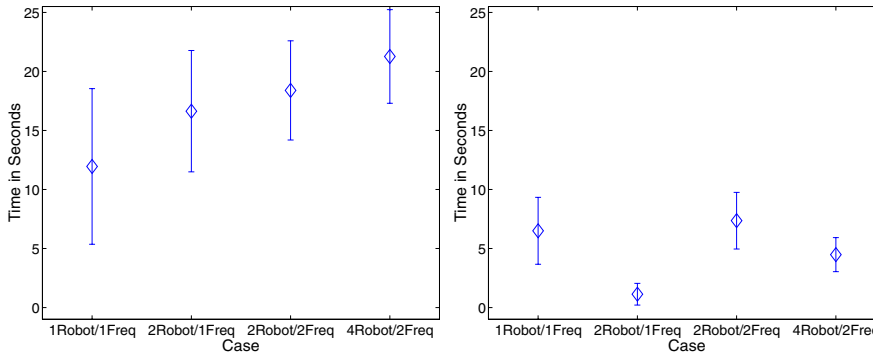


Fig. 9. Detect Motion experiment. On the left, the time (in seconds) the target was within the field of view of the Scouts. On the right, the time (in seconds) the target was actually viewed. This is calculated as the amount of time the target was in the field of view of a Scout even if the Scout wasn't active at the time

Figure 8 shows the total area in the field of view of the Scouts in each of the four cases. The area viewed in the four Scout case was significantly greater (at the 95% confidence level) than the areas viewed in the other cases, but not by a factor of four over that viewed by one Scout nor by a factor of two over that viewed by two Scouts. The size and configuration of the environment was such that there was usually a great deal of overlap in the areas viewed by individual Scouts. Redundancy was probably not as useful in this environment (two or three Scouts might have sufficed), but would probably be more effective in larger or more segmented environments.

The area traversed by the Pioneer that was within the field of view of the Scouts and the amount of time the moving target was visible were different across experiments. This was caused by the fact that the Scouts did not always hide in the best positions for viewing the Pioneer. In some experiments, one Scout was facing the wall instead of facing the open area, and so it did not contribute to the detection task at all. In other cases, two Scouts were very close with viewing areas almost completely overlapping. Again, increasing the number of Scouts might not increase performance much but increases robustness.

Figure 8 and Figure 9 show respectively the area traversed by the Pioneer that was in the field of view of the Scouts and the time the Pioneer was in the field of view of the Scouts for the different experiments. This gives an indication of the complexity of the task. The smaller the area and the shorter the time, the smaller is the opportunity for the Scout(s) to detect the Pioneer even when there is no swapping because a single frequency is used. The figures also illustrate the advantages of using a larger number of Scouts. Both the viewable area traversed by the Pioneer and the time that the Pioneer was in view have higher means and smaller variances when more Scouts were used. This provides a justification for the use of more Scouts than strictly needed to cover the area. Given the chance

the Scouts will not hide in good places, using more Scouts reduces the variability in the results and provides more opportunities for the detection of motion.

However, we should caution that the differences were not always statistically significant at the 95% confidence level. In particular, four robots were found to be significantly better than one in these measures but four robots were not found to be significantly better than two on different frequencies for either measure. Also two robots on the same frequency were not found to be significantly better than one for Pioneer path area viewed. This is due to the overall better placement of two Scouts using two different frequencies than two Scouts on the same frequency.

5 Related Work

Automatic security and surveillance systems using cameras and other sensors are becoming more common. These typically use sensors in fixed locations, either connected ad hoc or through the shared communication lines of "intelligent buildings" [10] or by wireless communications in "sensor networks" [8,10,15]. These may be portable to allow for rapid deployment [14] but still require human intervention to reposition when necessary. This shortcoming is exacerbated in cases in which the surveillance team does not have full control of the area to be investigated. Static sensors have another disadvantage–they do not provide adaptability to changes in the environment or in the task. In case of poor data quality, for instance, we might want the agent to move closer to its target in order to sense it better.

Mobile robotics can overcome these problems by giving the sensor wheels and autonomy. Robotics research for security applications has traditionally focused on single, large, independent robots designed to replace a single human security guard making rounds. Such systems are now available commercially and are in place, for example, in factory, warehouse, and hospital settings [12]. However, a single mobile agent is unable to be in many places at once—one of the reasons why security systems were initially developed. Further, these types of robots may be too large to explore tight areas.

Multiple robots often can do tasks that a single robot would not be able to do or can do them faster [3,20]. Most existing multi-robot systems have sufficient computing power on board. Energy consumption is a major problem [15] for small robots as well as sensors used in sensor networks. Due to their small size and limited power, most miniature robots have to use proxy processing [11] and communicate via a wireless link with the unit where the computation is done. This becomes a problem when the bandwidth is limited, as in the case of our Scout robots. Because of their limited size, not only is all processing for the Scout done off-board but the communication is also done on only a few communications channels. This severely limits the ability to control multiple robots at once.

The importance of communication and the impact that communication constraints have has been a subject of much research. Earlier work [2] has studied the impact of various types of communication on performance of multiple robots on a particular suite of tasks. The work was performed only in simulation and did not address the limitations of bandwidth we have with the Scouts. The

problem of deciding what to communicate is formulated in [21] as a decentralized Markov decision process. The objective is for the agents to find an optimal policy which will achieve the best global reward. In [6] a framework is proposed for coordinating the activities of multiple agents in the presence of unreliable and low-bandwidth communication. The idea is for the agents to plan how to best use the limited communication available. The work, which is still in the preliminary stages, would be quite relevant for our application.

The motion detection problem we have presented is similar to the Art Gallery problem [13,4], in which a robot attempts to find a minimal number of observation points allowing it to survey a complex environment. Our problem is complicated by the fact that the Scouts have a limited field of view, and that incidence and range constraints significantly affect their ability to detect motion. In [9], a randomized algorithm for sensor placement is proposed, which takes incidence and range constraints into account, but not the field of view.

More importantly, we are interested in detecting motion, not just in covering an area. The peculiarities of our motion detection algorithm combined with the limited field of view of the Scouts make detection of motion much more complicated. In addition, we are not free to place the Scouts in their best viewing position—they have to find a hiding place autonomously. Finally, since Scouts cannot place themselves in open areas, where they are likely to be seen and/or stepped on, the size of the environments they can cover is limited by the maximum distance at which they can detect motion.

The problem we addressed in this paper is assessing the effects of limited communication bandwidth on performance. Resource allocation and dynamic scheduling are essential to ensure robust execution. Our work focuses on dynamic allocation of resources at execution time, as opposed to analyzing resource requests off-line, as in [1,7], or modifying the plans when requests cannot be satisfied. Our approach is specially suited to unpredictable environments, where resources are allocated in a dynamic way that cannot be predicted in advance. We rely on the wide body of algorithms that exists in the area of real-time scheduling [18] and load balancing [5].

We have not considered other factors that could affect performance, such as knowledge of the motion of the moving object(s). Even though in our experiments we have used a single object moving at constant speed on a straight line, we do not use any of this information in the motion detection algorithm.

In earlier work [17], a simpler experiment was run in which only two Scouts were used to detect motion along a corridor. In that set of experiments, both Scouts were on the same video frequency, the Scouts did not have overlapping views and a Pioneer 1 mobile robot was used as the target. Several different experimental cases were run. In the first case, only the two Scout robots were used. In the second case, a third Scout robot using the same frequency as the first two was activated. However, instead of helping the first two with the task, the third robot simply used the bandwidth without contributing (as if it were involved in a different task). In the third case, the bandwidth had to be shared with two additional Scouts doing a different task, and in the fourth case, three superfluous Scouts were used. The ability of the first two Scouts to detect the target motion dropped from 80% in the best case down to 75%, 60% and 50%, respectively,

for the other experimental cases. Comparing those results with results described in this work illustrates how the placement of the Scouts in their environment, the shape and size of the environment, and the nature of the moving target can greatly affect the overall performance of the robotic team.

6 Summary and Future Work

Visual behaviors for autonomous operations of a group of Scout robots have been presented. Experimental results illustrating the ability of the Scout to position itself in a location ideal for detecting motion and the ability to detect motion have been shown. The software architecture we developed dynamically schedules access to physical resources, such as communication channels and framegrabbers, allowing them to be shared by multiple robots.

We have demonstrated how the communications bottleneck affects the overall performance of the robots. The next step is to add more intelligence into the behaviors to allow them to dynamically adjust their requested runtimes to react to the situation. We are also examining other kinds of RF communications hardware to increase the number of video channels, which is the main limiting factor in our system. The difficulty lies in the Scout's extremely small size and power supply. Very few transmitters exist that meet the requirements of our hardware.

Our scheduler currently does not take into account any high-level information about the kinds of tasks that the ARCs are attempting to accomplish. The purpose of the scheduler is to serve as a low-level transport layer which attempts to best distribute the load of (possibly multiple simultaneous) robot missions across the available hardware resources. Incorporating domain-level information into the scheduler will require a great deal more environmental information than is currently available. For instance, currently, the robots are not aware of the other robots when they are positioning themselves. Clearly, if they could determine their relative locations, they would be able to position themselves in a more effective manner. Additionally, if the robots were aware of their locations as well as the locations of their fellows in their environment and they knew the layout of the environment, they would be able to automatically adjust the scheduling algorithm to minimize the number of blind spots at any one time. These results would require that the robots have the ability to build maps and localize themselves within those maps. We are currently exploring methods for doing this with the Scout's camera and fusing information from multiple viewpoints.

References

1. E.M. Atkins, T.F. Abdelzaher, K.G. Shin, and E.H. Durfee. Planning and resource allocation for hard real-time, fault-tolerant plan execution. *Autonomous Agents and Multi-Agent Systems*, 4(1/2):57–78, Mar. 2001.
2. T. Balch and R.C. Arkin. Communication in reactive multiagent robotic systems. *Autonomous Robots*, 1(1):27–52, 1994.
3. Y. Cao, A. Fukunaga, and A. Kahng. Cooperative mobile robotics: Antecedents and directions. *Autonomous Robots*, 4(1):7–27, 1997.

4. V. Chvátal. A combinatorial theorem in plane geometry. *J. Combin. Th.*, 18:39–41, 1975.
5. G. Cybenko. Dynamic load balancing for distributed memory multiprocessors. *Journal of Parallel Distributed Computing*, 7(2):279–301, 1989.
6. M. desJardins, K. Myers, D. Morley, and M. Wolverton. Research summary: Communication-sensitive decision making in multi-agent, real-time environments. In *AAAI Spring Symposium on Robust Autonomy*, Mar. 2001.
7. E.H. Durfee. Distributed continual planning for unmanned ground vehicle teams. *AI Magazine*, 20(4):55–61, 1999.
8. D. Estrin, D. Culler, K. Pister, and G. Sukhatme. Instrumenting the physical world with pervasive networks. *IEEE Pervasive Computing*, 1(1):59–69, 2002.
9. H. González-Banos and J.-C. Latombe. A randomized art-gallery algorithm for sensor placement. In *Proc. ACM Symposium on Computational Geometry*, 2001.
10. B. Horling, R. Vincent, R. Mailler, J. Shen, R. Becker, K. Rawlins, and V. Lesser. Distributed sensor network for real time tracking. In *Proc. of the Int'l Conf. on Autonomous Agents*, June 2001.
11. M. Inaba, S. Kagami, F. Kanechiro, K. Takeda, O. Tetsushi, and H. Inoue. Vision-based adaptive and interactive behaviors in mechanical animals using the remote-brained approach. *Robotics and Autonomous Systems*, 17:35–52, 1996.
12. A. Kochan. HelpMate to ease hospital delivery and collection tasks, and assist with security. *Industrial Robot*, 24(3):226–228, 1997.
13. J. O'Rourke. *Art Gallery Theorems and Algorithms*. Oxford University Press, Aug. 1987.
14. D. Pritchard, R. White, D. Adams, E. Krause, E. Fox, M. Ladd, R. Heintzleman, P. Sprauer, and J. MacEachin. Test and evaluation of panoramic imaging security sensor for force protection and facility security. In *IEEE Int'l Carnahan Conf. on Security Technology*, pages 190–195, Alexandria, VA, Oct. 1998. Larry D. Sanson, ed.
15. V. Raghunathan, C. Schurgers, S. Park, and M.B. Srivastava. Energy-aware wireless microsensor networks. *IEEE Signal Processing Magazine*, 19(2), Mar. 2002.
16. P.E. Rybski, N. Papanikolopoulos, S.A. Stoeter, D.G. Krantz, K.B. Yesin, M. Gini, R. Voyles, D. F. Hougen, B. Nelson, and M. D. Erickson. Enlisting rangers and scouts for reconnaissance and surveillance. *IEEE Robotics and Automation Magazine*, 7(4):14–24, Dec. 2000.
17. P.E. Rybski, S.A. Stoeter, M. Gini, D.F. Hougen, and N. Papanikolopoulos. Effects of limited bandwidth communications channels on the control of multiple robots. In *Proc. of the IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems*, pages 369–374, Hawaii, USA, Oct. 2001.
18. J. Stankovic, M. Spuri, K. Ramamritham, and G. Buttazzo. *Deadline Scheduling For Real-Time Systems: EDF and Related Algorithms*. Kluwer Academic Publishers, Boston, 1998.
19. S.A. Stoeter, P.E. Rybski, M.D. Erickson, M. Gini, D.F. Hougen, D.G. Krantz, N. Papanikolopoulos, and M. Wyman. A robot team for exploration and surveillance: Design and architecture. In *Proc. of the Int'l Conf. on Intelligent Autonomous Systems*, pages 767–774, Venice, Italy, July 2000.
20. C. Weisbin, J. Blitch, D. Lavery, E. Krotkov, C. Shoemaker, L. Matthies, and G. Rodriguez. Miniature robots for space and military missions. *IEEE Robotics and Automation Magazine*, 6(3):9–18, Sept. 1999.
21. P. Xuan, V. Lesser, and S. Zilberstein. Communication decisions in multi-agent cooperation: Model and experiments. In *Proc. of the Int'l Conf. on Autonomous Agents*, pages 616–623, 2001.

Use of Cognitive Robotics Logic in a Double Helix Architecture for Autonomous Systems

Erik Sandewall

Department of Computer and Information Science
Linköping University, S-58183 Linköping, Sweden
erisa@ida.liu.se

Abstract. This paper addresses the two-way relation between the architecture for cognitive robots on one hand, and a logic of action and change that is adapted to the needs of such robots on the other hand. The relation goes both ways: the logic is used within the architecture, but we also propose that an abstract model of the cognitive robot architecture shall be used for defining the semantics of the logic.
For this purpose, we describe a novel architecture called the *Double Helix Architecture* which, unlike earlier proposals, emphasizes a precise account of the metric discrete timeline and the computational processes that take place along that timeline. The computational model of the Double Helix Architecture corresponds to the semantics of the logic being used, namely the author's Cognitive Robotics Logic which is based on the 'Features and Fluents' theory.

1 Introduction

This paper addresses the two-way relation between the architecture for cognitive robots on one hand, and a logic of action and change that is adapted to the needs of such robots on the other hand. The relation goes both ways. There is widespread agreement that a logic-based deliberative system is one necessary part of the robot architecture, but in addition we propose that an abstract model of the cognitive robot architecture should be applied to defining the semantics of the logic being used. Essential notions in the logic of the cognitive robot, such as the concepts of 'state' and 'action', the success or failure of actions, and even the notion of time within which observations are made and actions are performed - all of these notions are pertinent for both the logic and the system architecture.

1.1 Logic and Architecture

The logic being used here is the author's Cognitive Robotics Logic (CRL) [20] which is based on his earlier work on 'Features and Fluents' [17]. CRL is closely related to Doherty's Time and Action Logic (TAL) [4] and, somewhat more remotely, to the modern event calculus [23]. It is characterized by the use of explicit, metric time that allows for concurrent actions, actions of extended,

M. Beetz et al. (Eds.): Plan-Based Control of Robotic Agents, LNAI 2466, pp. 226–248, 2002.

overlapping duration, combinations of continuous and discrete change, characterizing the range of precision in sensors and actuators, and more. The approach presented here can probably be easily transferred to other logics in the same group. It appears that it can not easily be transferred to logics without metric time, such as the situation calculus [8], since the modeling of low-level, real-time processes is an important part of our enterprise.

We also present a novel architecture, called the Double Helix Architecture (DHA) that differs from previously proposed robotic architectures by being much more specific with respect to the time axis and its associated computational processes. It is common to define robot architectures in terms of graphs consisting of boxes and arrows, where the boxes denote computational processes and data stores, and the arrows denote data flow or control flow. Such diagrams abstract away the passage of time, which means that they are not sufficient for characterizing the semantics of a logic of time, actions, and change. The DHA will therefore be presented using two complementary perspectives, including both the traditional data-flow diagrams and the new time-axis diagrams.

The actual Double Helix Architecture contains more details than can be described in an article of the present size, so our account must be limited to the most salient aspects.

1.2 Implementation

The work on the Double Helix Architecture is a separate and rather small part of the WITAS project which aims at the construction of an intelligent helicopter UAV (Unmanned Aerial Vehicle)[1] [3] as well as research on a number of related technologies. The on-going and very large implementation effort for the on-board system in the WITAS helicopter is described in [5]. The DHA is a concept study and an experimental implementation; we wish to make it clear that the implementation of DHA is not integrated in the main implementation effort in the project. Also, similarities in design between the main WITAS system and the DHA are due to common background, so the design described here should not be interpreted as a description of the main WITAS system.

The priority for the DHA study is to obtain a design and an implementation that are as simple as possible, even at the expense of considerable idealization, in order to make it possible to establish a strong relation between the design and the corresponding logic. In addition, the DHA implementation provides a simulated environment of UAV and ground phenomena that is used for the continued development of a user dialogue system for WITAS[7]. It will be further described in section 9.

2 Assumptions

Since DHA is presently tried out in a simulated environment, it is very important to make precise what are the assumptions on the forthcoming situations where

[1] `http://www.ida.liu.se/ext/witas/`.

the system will serve an actual robot. Every simulation must be a simulation *of* something concrete.

2.1 Assumptions on the Robotic System

We focus on cognitive robotic systems with the following characteristics:

- They control mechanical robots (rather than e.g. 'web robots') and in particular, vehicles using computer vision as one important sensor and source of information about its environment.
- They have strict real-time requirements on them, but different aspects of the behavior operate on different time-scales, which makes it appropriate to use a layered architecture. In the case of our UAV application, it takes after all a while to fly from point A to point B, so there is room for computations that may take many seconds or even several minutes, if need be, besides other processes that operate on fractions of seconds as you would expect in an aircraft.
- Besides the robotic vehicle itself, which typically has a fixed set of components, the robot must also be able to represent a set of observed objects that changes over time, including for example roads, automobiles, and buildings on the ground (in the case of the UAV). Each such observed object is assumed to be within the robot's field of vision for a limited period of time, during which its properties can be identified although with limited precision and reliability. Dealing with such observed objects, their characteristics, and the events in which they are involved is a strict requirement on both the architecture and the logic.
- The robot must be able to communicate with persons or with other robots in order to give and receive knowledge about itself and about observed objects. This means, in particular, that its knowledge representation must support the way human operators wish to understand and talk about observed objects and their characteristics and processes, as well as of course the actions of the robot itself. Furthermore, this consideration suggests that the chosen logic for cognitive robotics should be able to represent the partial knowledge of each of several agents within one body of propositions.
- In particular, the actions of the robot should not only be represented as simple actions, such as 'fly to position (x, y, z) and hover there'. The operator may also e.g. wish to inquire about how the flight has been performed or will be performed, or impose restrictions on trajectory, velocity, altitude, etc.

There are a number of other requirements that must be made in the case of the WITAS application, or any other concrete application, but the present list will be sufficient for defining the direction of the present work.

2.2 Assumptions on the Deliberative System

We focus on a deliberative system that is capable of the following functions, all of which refer to the use of actions and plans:

- Execute a previously defined plan, taken from a plan library, with consideration of real-time constraints. From the point of view of the cognitive system, 'executing a plan' means to communicate the plan, piecemeal or in a single shot, to the robot control system and to receive the corresponding feedback from it.
- Generate a plan for achieving a given goal.
- Interleave planning and plan execution, again taking real-time constraints into account. Planning consumes real time.
- Engage in dialogue with one or more users, which also may involve the use of, and the making of plans for dialogue actions. Communicate its current plans and the motivation for its past, current, and planned actions, in dialogue with these users or operators.
- As a long-term direction, the system should also be able to 'learn by being told', that is, to receive high-level advise about how it may modify its own behavior.

We also emphasize crisp, true/false distinctions rather than graded certainty. Some of the constructs described here will rely on uncertainty management on lower levels of architecture, but we assume that this is a separate issue. This is natural in particular because of the priority that is made on the dialogue aspect.

3 Related Work

The topic of 'robotic architecture' recurs very frequently in the literature about A.I. robotics. The concept of architecture itself is not very precise. Often, such as in Dean and Wellman's book on Planning and Control [2], an architecture is seen as a prescription for how a number of modules or subsystems are to be assembled. Architectures in this sense are typically characterized using block diagrams, as was mentioned above. Others, such as Russell and Norvig in their standard textbook [14], view an architecture as a computational infrastructure that can be 'programmed' to perform the desired cognitive tasks. In this case the architecture may include both the robotic hardware, its software drivers, and implementations of specialized languages such as RAPS [6] that provide a high-level platform for building applications.

Our use of the term 'architecture' in this article is closer to the one of Russell and Norvig than the one of Dean and Wellman inasmuch as we emphasize the importance of being able to 'program' a generic software system by attaching and replacing software modules (plug-ins), written both in general programming languages and in specialized languages including, but not restricted to, logic.

In yet another interpretation, the word 'architecture' refers in a comprehensive way to all the significant design choices and facilities on all levels of a complex system. That is not the way we use the term here.

The notion of multi-level architectures gained acceptance within A.I. during the 1980's [1,22], superseding earlier and often simpler notions of a sense-deliberate-act cycle. Robotic architectures in modern A.I. usually emphasize

the need for the different layers to operate concurrently, so that low-level and intermediate-level sensori-motoric processes can proceed while high-level deliberation is allowed to take its time. Still there is not much conceptual integration between the different levels; the procedural languages that are often proposed for use on the higher levels of these systems do not usually have the expressivity that is required e.g. for characterizing the precision of sensors, or for specifying and adding to control laws. Please refer to [10] and to Chapter 2 in [11] for surveys of architectures for intelligent robots.

The extensive research on logics of actions and change includes several approaches that attempt to bridge the gap between the 'lower' and the 'higher' levels of robotic performance. This includes in particular the work by Rao, Georgeff et al.on BDI architectures [13], the work by Reiter, Levesque, et al.on GOLOG [9], Shanahan's work on robot control using the modern event calculus [23], and Nilsson's triple-tower architecture [12]. (The related topic of planning domain description languages is also potentially relevant in this context, although action planning in A.I. has until recently tended to stay away from the challenges of cognitive robotics). However, none of the mentioned approaches contributes to the clarification of the semantics of actions in terms of more elementary principles, which is a major concern of the present work. Also, with the possible exception of Nilsson's work, in all of these approaches the architecture seems to be chosen as an implementation of the logic being used. In our work we wish to see the architecture as a way of orchestrating the cohabitation between deliberative processes and conventional, procedural computation.

Two aspects of our own earlier work are used for the Double Helix Architecture and the associated logic. With respect to the representation of continuous change, we proposed in 1989 an approach to embedding differential equations in a logic of actions and change [15,**?**]. This work was later extended for representing the distinction between actual values of state variables, observed values, and set values, and for the representation of criteria for the success and failure of actions [18]. These representational methods have been included in Cognitive Robotics Logic.

Secondly, we proposed in [19] a logic characterization of goal-directed behavior, which was defined as behavior where given goals cause an agent to select one of several possible plans for achieving the goal, to attempt to execute the plan, and to try again (possibly with the same plan, possibly another one) if the first attempt fails. This is a standard aspect of rational behavior. Other aspects of rational behavior such as the use of utility measures in the choice of plans was not represented in that work, and is currently a topic for research.

4 The Double Helix Architecture

The present implementations of DHA, i.e. DOSAR and DORP, are embedded logic systems. In their overall architecture there are several specific uses of logic formulas and of deduction and other inference operations. Logic formulas occur both as a kind of data structures or messages (typically for ground literals), and

as rules that are used for forward inference from observations, or for answering queries deductively.

We shall describe the DHA in two steps. The first step corresponds to a variant of a three-layer architecture where the intermediate layer is minimal; the second step corresponds to a full three-layer architecture.

4.1 The Elementary Double Helix Architecture

Figure 1 shows a simplified structure of the computational processes in the system along a time axis that runs vertically and from top to bottom. The system has a cyclic behavior where each cycle consists of a relatively long *evolution and deliberation phase* and a shorter *information exchange phase*. The 'evolution' here refers to the spontaneous developments in the physical world within which the robot or (in our case) the UAV operates, together with the automatic control processes where the robot is engaged. Concurrently with it, and independently of it, there is the 'deliberation' where the system generates and checks out plans and predicts the likely immediate future for itself and in its environment.

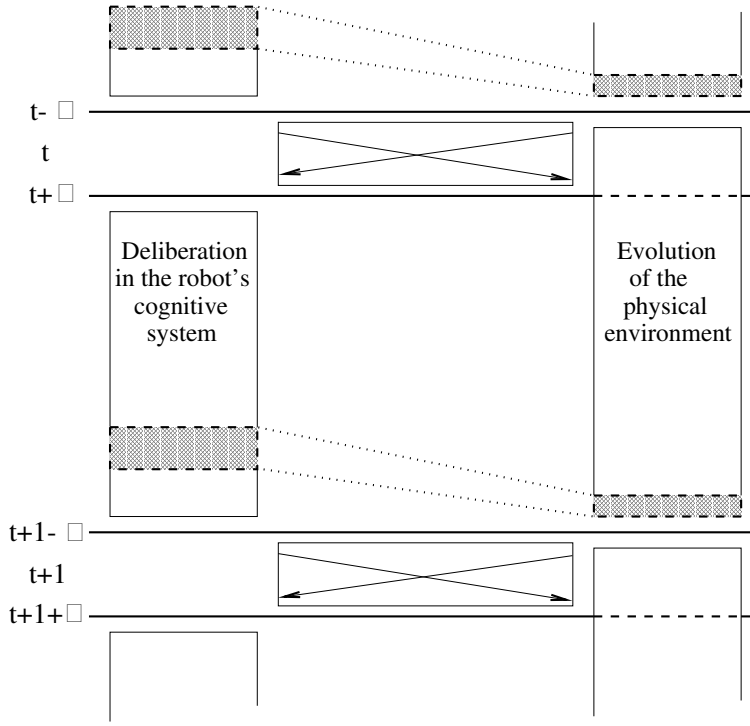


Fig. 1. View of the Double Helix Architecture along the time axis. Shaded areas refer to user intervention (see section 8)

For the purpose of developing the higher levels of the architecture, it is appropriate to replace large parts of the evolution subsystem by a simulation, provided that the simulation is a sufficiently good imitation of the target system *in those aspects that are relevant for the higher levels that are being developed*. Systematic use of simulation in this way is standard practice in concurrent engineering, but it requires of course that the relevance of the simulation is carefully monitored and evaluated.

The information exchange phase occurs each time it is appropriate to deliver sensor data, and observations based on them, to the deliberation system. At the same time, the deliberation system may update its decisions on ongoing actions or activities: helicopter maneuvers, camera movement, method for communication with the ground, etc. We expect that the information exchange phase will occur with fixed frequency most of the time, although the frequency may be reset corresponding to different flight modes and the system should be able to accommodate occasional deviations from the fixed frequency.

The data flow through this structure looks like a modified double helix: the evolution strand may lead to an observation that is transmitted to the deliberation system at time t, causing it to deliberate and to initiate an action at time $t+1$, whereby the evolution from time $t+1$ onwards is affected. The modification to the double helix occurs because both the evolution and the deliberation strand has a persistence of its own, besides receiving impulses from the other side. It is for this reason that we refer to this design as a Double Helix Architecture (figure 2).

Another and more conventional projection of the data flow is shown in figure 3, where the time dimension has been removed (it is orthogonal to the paper or screen), and one can distinctly see the cycle of data flow between the evolution line, represented to the right in the figure, and the deliberation line to the left. Atomic CRL formulas of the following kinds are used for transmitting information between the different subsystems:

- $H(t, obs(f), v)$ expressing that the value of the feature f is estimated as v based on observations at time t
- $H(t, set(f), v)$ expressing that the controllable feature f (e.g. the desired forward horizontal acceleration of the UAV) is set to v at time t by the procedure carrying out the current action selected by the deliberation system
- $D(s, t, a)$ expressing that the action a started at time s and terminated at time t
- $Dc(s, t, a)$ expressing that the action a started at time s and is still going on at time t. This is an abbreviation for $\exists u[D(s, u, a) \wedge t \leq u]$ where the c in Dc stands for 'continuing'
- $H(t, fail(a), T)$ expressing that $D(s, t, a)$ and the action failed. (The execution of an action is classified in a binary way as success or failure)

These are the atomic formulas that are handled by the deliberation system and by the high-level action execution system. Formulas of the form $H(t, f, v)$ are also in the logic, and designate that the true value of the feature f at time t is v.

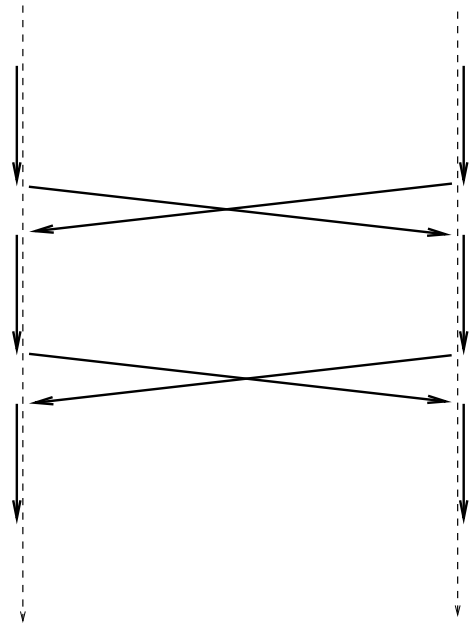


Fig. 2. The Double Helix of the Architecture

This will of course not be directly known to the system, but the logic may make reference to such expressions in contexts such as $H(t, f, v) \wedge H(t, obs(f), v') \rightarrow abs(v - v') \leq \varepsilon$ which expresses that the observation error for the feature f is less than a known ε. In this way it is logically possible for the system to *reason about* the value of v in $H(t, f, v)$ although its exact magnitude is not known.

High-level and low-level action execution are handled differently in this design. As is shown in figure 3, there must necessarily be a number of feedback control loops that operate with a higher frequency, and whose detailed operation need not make active use of the logic. For example, one action may specify the set value for the forward acceleration of the UAV, in the course of high-level action execution. The feedback control may then use a 'controlled state variable' for the level of the throttle at each instant.

The feature expressions f can be e.g. $pos(car4)$, meaning the current position of an object identified as car number 4, or $vel(car4)$ for its forward velocity, or again $acc(car4)$ for its forward acceleration. Observed instantaneous events are represented in the same way but often with boolean values. For example, an instantaneous event where $car2$ yields to $car4$ may be represented as $H(t, yield(car2, car4), T)$ where the capital T stands for reified truth-value. In such cases $H(t, obs(yield(car2, car4)), T)$ designates that the event recognizer has reported such an event; it may or may not have actually taken place. (Here we assume, for the sake of simplicity, that there is an objective definition of

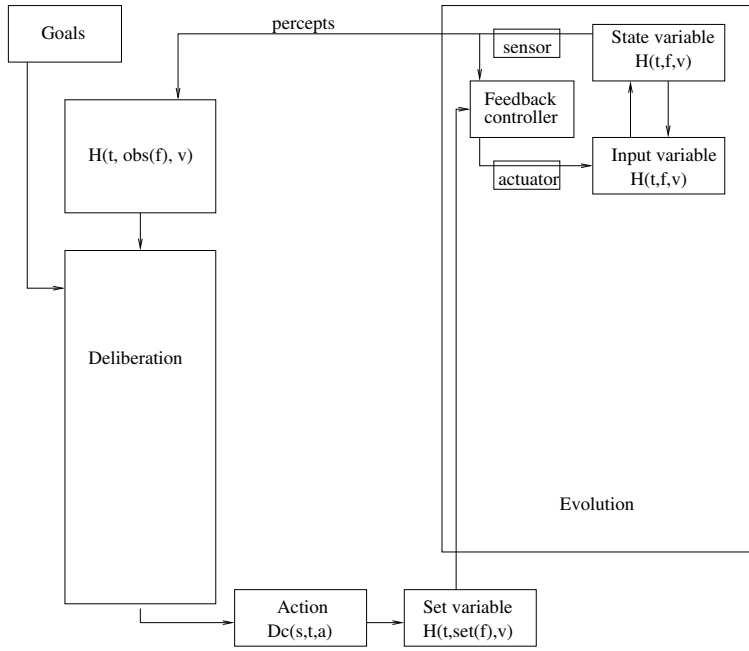


Fig. 3. Dataflow-oriented view of the elementary Double Helix Architecture

whether that event has taken place in terms of the actual trajectories of the cars being referenced).

The deliberation and evolution boxes in figure 3 represent ongoing activities that interact in each information exchange phase, but which also have their own continuity. Observations in the form of formulas $H(t, obs(f), v)$ do not persist intrinsically. For elementary features f they have to be calculated and reported anew in each cycle. The same applies for set-value statements of the form $H(t, set(f), v)$ which have to be recalculated in each cycle by the control procedures for the actions that are presently being carried out.

The expressions denoting actions do have persistence, however. When the deliberation system decides to initiate an action a at time s, it does so by asserting the formula $Dc(s, s+1, a)$, meaning that the action started at time s and is still executing at time $s+1$. This formula persists as $Dc(s, t, a)$ for successively incremented t, and for each timepoint t it is used to infer the appropriate values for 'its' set state variables or features.

There are several ways that an action can terminate: by a decision of its own control procedure, or by the decision of the event recognizer, or by deliberation. Also the termination may be with success or failure. In any case, however, the termination is done by adding a formula $D(s, t, a)$ saying that the action a started at time s (which was known before) but also that it ended at time t. Furthermore in the case of failure the formula $H(t, fail(a), T)$ is also added. In case of success

the formula $H(t, fail(a), F)$ may be added explicitly or be obtained by default; this is an implementation consideration.

The persistence of formulas of the form $Dc(s, t, a)$ is therefore disabled by the appearance of the $D(s, t, a)$ formula. This *action persistence mechanism* in its simplest form can be characterized by the non-monotonic formula $Dc(s, t, a) \wedge Unless\ D(s, t, a) \rightarrow Dc(s, t + 1, a)$ An extended set of rules is used for also dealing with goal-directed composite actions, where the conclusion of one action with success or with failure may initiate additional actions.

4.2 Event Recognition in the Full Double Helix Architecture

If taken literally, the Elementary DHA would require observations to be fed from evolution to deliberation at every time-step. This is however not computationally realistic, and figure 4 shows the full architecture also containing a non-trivial intermediate layer. (Note that the usual, vertical ordering of the layers has been replaced by a horizontal one in figures 3 and 4. This facilitates comparison with figure 1, where deliberation and evolution are also placed side by side). The box labeled 'event recognizer' represents observation aggregation processes whereby a flow of detailed observations are combined into higher-level concepts, such as the arrival of an observed object into the field of observation, or the beginning and end of activities where the observed object is involved.

The term 'event recognizer' covers a fairly broad range of computations, therefore. From a propriosensoric point of view for the robot, it can assume a fixed set of subsystems, sensors, and actuators, much like in an industrial process control system, and then the term event recognizer is accurate. For observed objects, on the other hand, the same computational process must both administrate the introduction and abandonment of data objects that represent observed physical objects, and the recognition of properties and events where these are involved.

In these respects, the 'event recognizer' is interdependent with the computer vision system. Each of them must adapt to the capability of the other one, and to the extent that they are developed independently each of them should specify what assumptions it makes about the other side. Consider, for example, a dynamic vision system that is able to segment incoming images obtaining more or less precise 'blobs', to track them reliably during certain intervals of time even when there are several of them at the same time in the field of vision, and that is also able to determine and to report whenever there is a significant risk that the presently observed blobs have been mixed up. This is a concrete and well defined situation for the event recognizer as well as for the deliberation subsystem.

The following is in outline how such a component of the architecture can be related to the cognitive robotics logic. When the event recognizer receives notice of a new 'blob' it generates a *tracker object* b and at least one *activity demon* that supervises to the tracker object. The tracker object keeps track of the continued evolution of the 'blob' as long as the underlying vision system is confident that the blob still refers to the same object. An activity demon of type p that is administrated by a perceiving agent a and that refers to a tracker object b that

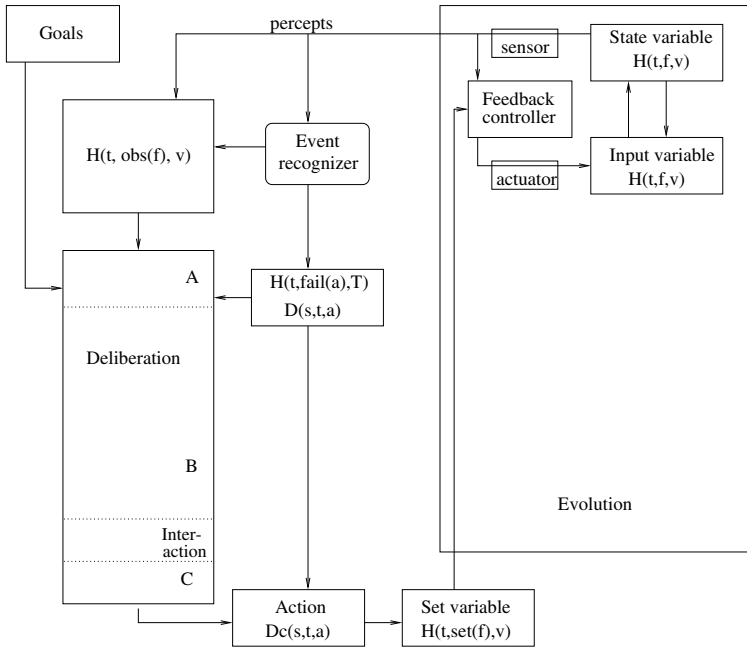


Fig. 4. Dataflow view of the full Double Helix Architecture, including the middle layer

is considered (possibly after deliberation) to designate an actual object c, serves to identify an activity $p(a, c)$ with a particular duration $[s, t]$ that is identified by the demon while using the information collected by the tracker object. In logic terms, the occurrence of this activity is represented as $D(s, t, p(a, c))$, and this is the formula that shall be made available to the deliberation part of the system if the mapping from b to c is certain. Notice in particular that the 'events' that are recognized in this way can have extended duration, and that we are not only using instantaneous events.

The current DOSAR system contains an explorative implementation of the computational chain from tracker objects via activity demons, to activities expressed in logic, but without using any actual vision system. (As stated in section 1.2, our implementation is not part of, or connected to the current D/R software architecture used with the actual WITAS UAV platform[5]). Instead, tracker objects are generated directly from the simulator. This is sufficient as a first step, but additional work is needed in order to better understand the logical aspects of perception in relation to realistic assumptions about what information can be obtained from vision systems and other high-level sensor systems.

4.3 Action Execution in the Full Double Helix Architecture

A cognitive robotic system must at least provide two levels for characterizing actions and plans, namely as procedures in the programming language being used, and as formulas in the chosen logic of actions and change. The DOSAR implementation of the full Double Helix Architecture offers the following intermediate levels of description as well:

- As programmable hybrid automata. If the logic refers to one action as for example $D(s, t, flyto(\textit{building-12}))$, where s is the starting-time and t is the ending-time of the action characterized by *flyto(building-12)*, then that action may be implemented as a hybrid automaton where the state transitions are understood as mode change within the action.
- As composite actions. The third argument of the predicate D may use composite actions such as $seq(a, b)$, $try(a, b)$, and $if(p, a, b)$ where a and b are again actions, recursively, and p is a condition corresponding to and combining the second and third argument of the predicate H. Here $seq(a, b)$ executes first a and then b if a was successful, whereas $try(a, b)$ executes a and then b if a failed.
- By pursuing an agenda. An agenda contains a set of intended forthcoming actions whose ramifications have been checked out by careful deliberation. The system is able to interpret such an agenda and to execute the actions in it successively. The generation process for this agenda will be described in section 7.

There is no major difference between the expressivity in these levels of action specification as long as one only considers normal, successful execution of actions. However, the later description methods are more powerful when it comes to characterizing and controlling action failure, in order to diagnose the fault or take alternative action. The similarity of primary expressivity is then an advantage: we expect to implement facilities for automatic conversion between the action description levels in order to 'compile' and 'decompile' action scripts when needed. Because of space limitations it is unfortunately not possible to describe the hybrid automata and composite actions levels here.

5 The Logic

5.1 The Use of Logical Inference

If the use of logical formulas in this architecture were restricted to their use as information units that are transmitted between the various computational processes, then it would be merely a choice of data structure. The use of logic becomes significant when one or more of those computational processes can be organized as deduction or other forms of inference, or if the specification of a process can be made in that way as a basis for verifying the implementation.

The deliberation process is the most obvious use of logic inference. We shall return to it in section 7, but it is not the only place for logical inference in

DHA. The 'middle layer' of the full DHA can be understood as a dynamic rule-based system, where rules react to messages obtained from the two main strands of the architecture, and where also the set of rules changes dynamically. Action persistence, the management of composite actions, and the management of basic goal-directed behavior can be performed using a fixed set of rules. At the same time, event and activity recognizers as well as hybrid-automata definitions of actions are in principle situation-action rules that apply for limited periods of time. In all these cases the rules can be expressed in CRL for the purpose of uniform specification, although the implementation looks different in the interest of computational efficiency.

Action persistence was characterized by a simple non-monotonic rule above, and that rule can be extended to also handling the successive invocation of actions in a linear script, and to the realization of goal-directed behavior. A strict application of the structure in figure 1 would imply that if an action fails then it is reported as discontinued, the corresponding set feature formulas $H(t, set(f), v)$ are no longer generated, but also the deliberation system is informed about the failure and is able to consider trying some other action towards the same goal, or even trying the same action again. However, this involves a delay, and one will often be interested in having a shorter path from the failure of one action to the initiation of a substitute action.

The formalization of goal-directed behavior in logic [19] can fill this need. That method uses a small number of axioms, operationally used for forward-inference, where the antecedents involve the kind of expressions that indicate the termination of actions in our present approach, that is, as $D(s, t, a)$ and $H(t, fail(a), T)$. These axioms, together with an appropriate plan library, serve to take the step from one action failure to the start of a substitute action without any intermediate deliberation cycle.

Furthermore, the implementation of specific actions can be done using rules of the general form $Dc(s, t, a) \wedge \neg D(s, t, a) \rightarrow H(t, set(f), v)$ with an opportunity to add additional conditions in order to control which features are to be set, and with what values.

5.2 DHA-Based Semantics

Almost all logics for action and change that have been proposed until now use integers, rather than a continuous domain as their concept of metric time. Surprisingly enough, however, there has been little consideration of the computational phenomena along that integer time axis. When combining the DHA and the CRL, we explicitly intend the time-steps in DHA to also be the time-steps in the integer time metric of the CRL.

The usual constructs in logics of action and change raise a number of definition issues, such as:

- Do we allow actions that take zero time, or must all actions take at least one time-step?

- Is it possible to invoke an action that is not applicable, and if so, does that lead to the action 'failing', or is it simply not semantically possible to invoke an action in a situation where it is inapplicable?
- Is it possible to have an action of a particular type from time $t1$ to time $t2$ and then another action of the same type from time $t2$ to time $t3$, or is it just one single action from $t1$ to $t3$ in such a case?

We propose that such questions ought not be resolved by vague notions of 'common sense' applied to particular scenario examples. Instead, they should be answered by defining the invocation, execution, termination, success, and failure of actions in terms of a precisely expressed agent architecture, and in particular the DHA. In doing so, one obtains a firm foundation for these concepts in themselves, and also for the subsequent project of defining high-level, discrete actions in terms of low-level, continuous or piecewise continuous phenomena.

As a topic of future research, we suggest that some aspects of symbol grounding, which is a topic of great current interest, can also be analyzed in this way.

6 The Temporal Situatedness of Logical Rules

The DHA time axis is important both for the logic and for the deliberative and other computational processes. In this section we shall argue that a precise understanding of the architecture's time axis is very useful when designing axiomatizations, deliberation systems, and function-specific software alike.

CRL predicates such as $H(t, f, v)$ and $D(s, t, a)$ treat time as some of the arguments, at a par with non-temporal arguments. From a computational point of view, however, it is natural to use the concept of a 'current state' that describes the state of the world at the 'current time'. Instead of placing all formulas in one single store and using the current time (as an integer) as a selector in that store, one may decide to collect all formulas of the form $H(t, f, v)$ and $Dc(s, t, a)$ with the same t into a heap of their own, which will be called a *state description* or simply a *state* for time t. Many parts of the computation can then be understood as performing a transformation from the state for one timepoint t, creating or amending a state for the timepoint $t + 1$. We use the terms 'old state' and 'new state' for those two states as seen from the computation. Note, however, that the considerations that follow apply as well if one uses current time as a selector into a global formula store.

The concurrent exchange that is illustrated by the two crossing arrows in the figures indicate the need for distinguishing old and new state, and not just doing assignments into one 'current' state. In this case the two states for information exchange at time t refer to the case 'just before' t and 'just after' t, respectively. This is indicated in the diagram as $t - \varepsilon$ and $t + \varepsilon$; if the information exchange is approximated as instantaneous then we are talking here of left and right limit values along the continuous time axis.

Other parts of the architecture relate in entirely different ways to the architectural time axis. For the computation in the deliberation subsystem, the 'old

state' refers to $t + \varepsilon$, and the 'new state' that is constructed in the course of deliberation refers to $t + 1 - \varepsilon$.

For the computation in the evolution subsystem, finally, the distinction between 'old' and 'new' state is probably not useful at all. There will normally be one 'current state' that is updated a number of times between the discrete timepoints that are here designated as t and $t + 1$.

The cycling of the states, where what has been set up as the 'new state' becomes the 'old state', and a new 'new state' is initiated, can in principle be made either just before, or just after the information exchange phase, since those are the only natural synchronization points. It is more natural to do the cycling just before, so that the information exchange phase can be defined to construct essential parts of its 'new state' from its 'old state'. Correspondingly, if current time is represented as a single variable *currenttime*, there will be an assignment of the form $currenttime := currenttime+1$ just before the information exchange phase. The location of this time shift is illustrated in figure 5 (time axis) and in figure 6 (cycle).

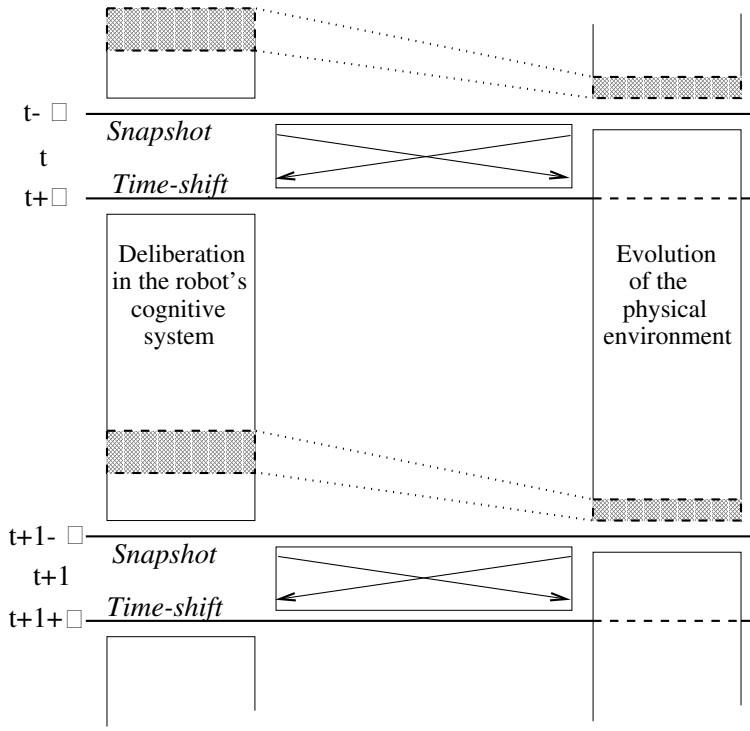


Fig. 5. Location of Timeshift and Snapshot in the time axis view of the Double Helix Architecture

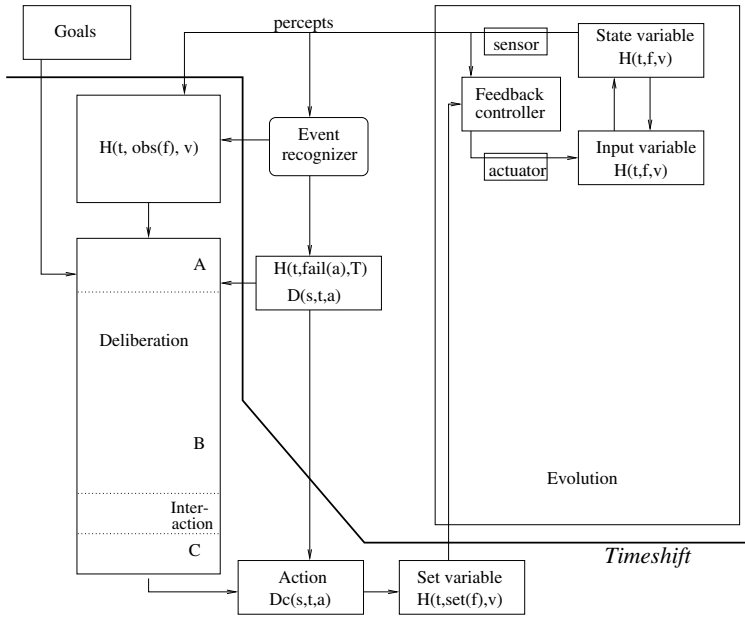


Fig. 6. Location of Timeshift in the dataflow view of the full Double Helix Architecture

In our experience it was difficult to write the logical rules for the different processes before we were clear about the difference between the various computational subsystems with respect to their temporal situatedness. The same need for a precise understanding of information states and their changes along the time axis was even stronger when it came to writing specialized software that was to be embedded in ("plugged into") the architecture.

7 Deliberation

7.1 Robotic Deliberation Using Natural Deduction

Let us turn now to the deliberation part of DHA, which uses a variant of natural deduction. The WITAS DORP system is an early and partial implementation of the following design.

Natural deduction is an inference system (set of inference rules) that allows one to set up and pursue 'blocks' or 'subproofs'. Blocks can be nested recursively. Each block starts with a *hypothesis* that can be selected as any well-formed formula, and proceeds with conclusions that may rely on the hypothesis and on any previous conclusion in any outer block. A block that begins with the hypothesis H and has a formula P on a later line supports a conclusion $H \rightarrow P$

outside the block. Special restrictions apply for the use of variables in block hypotheses. Please see e.g. [24] for an introduction to natural deduction.

This technique is useful in the design of a cognitive robot that needs to reason predictively about its immediate future and to do progressive planning, in particular if natural deduction is modified slightly as we shall describe here. Consider for example the situation where an agent has adopted a goal g, and it considers two alternative ways of achieving the goal, characterized by the actions $seq(a, b)$ and $seq(c, d, e)$, respectively, where seq is the sequential composition operator for actions, as was described in section 4.3. The deliberation system will then create two natural deduction blocks, one of which uses the initial hypothesis to execute the action $seq(a, b)$ starting at some timepoint greater than the present time, whereas the other one uses the initial hypothesis of executing $seq(c, d, e)$ similarly.

The system's deliberations have the effect of (a) adding conclusions, natural deduction style, to one or both of these blocks, based on the hypotheses in question and on known facts that are available 'outside' the block, and (b) strengthening the assumptions when this is needed in order to have an adequate plan. Strengthening the assumptions can be done e.g. by choosing a particular itinerary for a move action, or a particular object or tool for the action.

These deliberations serve to identify consequences of the proposed action in the physical and operational environment at hand. They also serve to derive detailed aspects of the actions, such as how to initiate and conclude the action. For example, in a sequence of actions each of which requires movement of the robot or some of its parts, the final phase of one action may be constrained by the requirements of the initial phase of the succeeding action.

7.2 Opening of Natural-Deduction Blocks

The formulas that are communicated in the operation of the double helix are consistently ground (i.e. variable-free) literals. The deduction in hypothetical blocks must be somewhat more general, however, since it is necessary to introduce variables for the starting-time and termination-time of actions that occur in the block-defining hypotheses. It is also sometimes necessary to introduce variables corresponding to the choices that need to be made, for example the choice of object or instrument for a particular action, and corresponding to feature values that are not yet known at the time of prediction or planning.

In all of these cases, however, the intention is that the variables that occur in a hypothetical block will later on be instantiated, either by the decision of the agent (starting time of actions, active choices), or by the arrival of information from observations (ending time of actions, in most cases, and feature-values that become known later).

In ordinary uses of natural deduction, free variables in the block hypotheses may be used for subsequent introduction of a universal quantifier on the implication that is extracted from the block, and after that for multiple instantiations of the all-quantified formula. In our case, however, we do not foresee more than one instantiation of a given block. Therefore, when an action or action-plan has

been selected after having been analyzed in a natural-deduction block, one can simply 'open up' the block and declare that all propositions in the block are in fact propositions on the top level (or in the case of nested blocks, on the level immediately outside the block being opened). At the same time, the free variables in the block's hypothesis become reclassified as constant symbols. Some propositions must also be added, in particular, a proposition specifying the actual starting time of the block's action, and propositions that clarify how the invocation of later actions depends on the success or failure of earlier actions in the block.

7.3 Self-Reference for Natural-Deduction Blocks

Besides investigating consequences of particular plans on a logical level, a rational robot must also be able to compare and evaluate the effects of alternative plans. In order to make such considerations accessible to the logic formalization, we introduce a concept of *block designator* and make the following technical adjustment of the approach that was described in the previous subsection. Each new block that represents a possible course of action is assigned its own, unique designator bd_n which is a constant or function symbol. A block for a particular plan e.g. $seq(a, b, c)$ is represented by introducing a block with $Decide(bd_n, s)$ as its hypothesis, and a new formula of the form $Decide(bd_n, s) \rightarrow \exists t.D(s, t, seq(a, b, c))$ outside that block. It is clear that formulas of that kind can be freely introduced without significantly extending the set of theorems. The consequence $\exists t.D(s, t, seq(a, b, c))$ follows trivially inside the block.

The more general case where an action has additional parameters can be handled by a straightforward generalization. Parameters whose value is chosen by the agent become arguments of the block designator; parameters that emerge from the development of the world obtain an existential quantification similar to the action's ending-time t.

In this way, the block designator can be used as a handle at which one can associate utility information for the actions defining the block, as well as procedural information about the resources required for the deliberations in the block and the chances of obtaining a desired result.

8 External Contributions and Logs

We have already explained how different parts of the DHA system relate differently to the time axis and the breakpoints along it. In the practical system there are a number of other things that also occur along the same time axis, and that likewise benefit from a clear account of the temporal relationship between different computational tasks.

When designing and testing the deliberation subsystem, including the rules that it contains, it is convenient to have one well-defined point in the cycle where the user can intervene in order to inspect the current contents of the 'old' and

the 'new' state, and also to change some of their contents. If the evolution is implemented as a simulation then one may need to intervene in a similar way in the simulation results. The shaded areas in figures 1 and 5 indicate the natural location for this interaction on the time axis.

Communication with a user via a language or multimedia interface, on the other hand, is better made as a part of the information exchange phase, and concurrently with the other exchanges there, so that the deliberative system can consider all its input at once in each cycle.

For the purpose of design and debugging, as well as for the tutorial purpose of understanding how the system works, it is also useful to produce a log of the current state of the system at some point in the cycle. We found it natural to let the log consist of such snapshots of the 'new state' just before information exchange phase, as indicated in figure 5 (time axis view) and figure 7 (dataflow view).

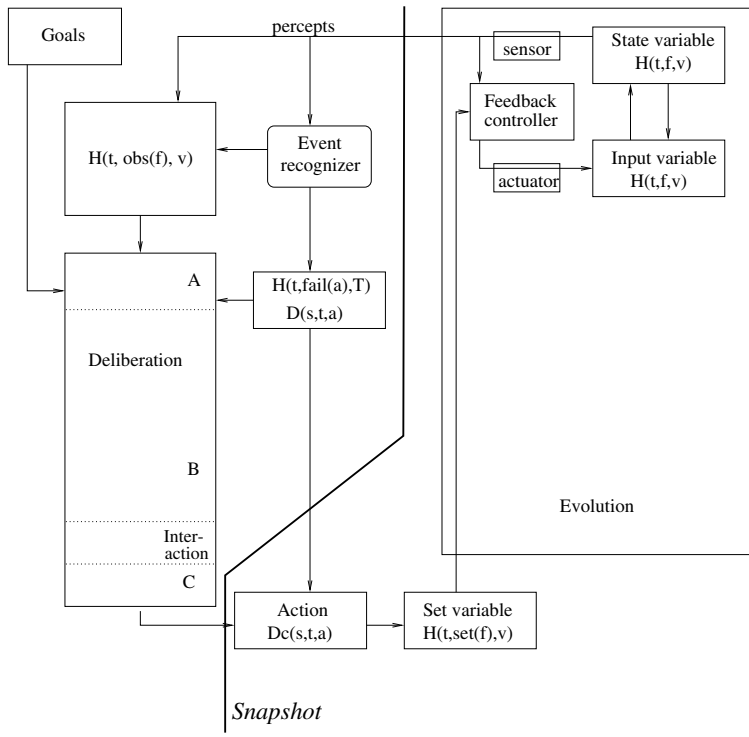


Fig. 7. Location of Snapshot in the dataflow view of the full Double Helix Architecture

For detailed design and checking, it is useful to have a single-step capability where one can easily intervene at each point in (simulated) time. In an interac-

tive and incremental programming system such as Lisp, it is natural to let the interaction (shaded segments) be done through the system's general command loop, and to define one interactively available operation that advances the system one cycle, that is, from one interaction point to the next one. This is yet another way that the DHA cycle can be split up in order to match the needs of a particular usage situation. The following sequence of operations is obtained for the single-step case, using the references A, B, and C of figures 6 and 7:

1. Deliberation, part C
2. Contribute snapshot to log
3. Information Exchange (two directions, performed concurrently)
4. Let 'new state' become 'old state', or advance timepoint counter by 1
5. Deliberation part A (interaction with operators), followed by Deliberation part B (deliberation proper); all of this concurrently with Evolution.

9 Implementation and Validation

The software implementation of the Double Helix Architecture serves two very different goals. Ideally, we would like to have an implementation that can both be used as a significant component of the actual on-board system in the WITAS helicopter, *and* at the same time is such a crisp and transparent computer program that it can be formally characterized by axioms in CRL with provable consistence between axiomatization and implementation.

Neither of those goals has been achieved yet, but we try to balance the priorities so that both can eventually be reached. A crisp, analyzable software that is not used on-board, but which is reasonably isomorphic to the actually used software according to an informal analysis, would also be an acceptable final result.

The presently working DOSAR system contains a simulator where a simulated helicopter flies over simulated cars that move around in a small street network with an update frequency of 2.5 Hz. A graphics package allows to display the ongoing simulation in real time. Helicopter actions can be defined on the three levels that were described in section 4.3, and the present, small library of helicopter actions is being extended gradually. On the perception side, a few recognizers for car activities have been written. They interface to the logic level of the system in the way that was described in section 4.2.

DOSAR also has the capability to represent a catalogue of scripts that can be used as methods for achieving goals. Each script is represented as a composite action expression. On command, the system can create natural deduction blocks (subproofs) for each of the applicable methods for a given goal, and construct the corresponding elementary propositions in a way that is local to the block. On another command it can execute the actions in a given such block, which means that the 'decision' is presently left to the operator. The program parts for inference of additional consequences within a block, or in combinations of blocks are next in line for realization. The system presently relies on a library of plans, and does not at present provide for planning in the classical sense.

The decision what to include and what not to include in the present implementation was partly guided by the theoretical interests, and partly by the immediate practical use of the software. In the short term, it is to be used as the simulation counterpart for the continued development of the WITAS high-level operator dialogue facility that is being built for us at Stanford University [7]. This consideration dictated a priority on simulating car movements and arranging for logic-level observations on them, as well as the use of a plan library rather than autonomous planning, and the choice of a quite crude simulator for helicopter movements.

Later on, it is intended to use DOSAR as the bridge between the dialogue and on-board systems, at which point the simulation part will be removed and replaced by an interface to the on-board system. (This interface will also need to include a safety-switch under operator control). Different approaches to deliberation are presently being pursued in the on-board system for WITAS as well as in the DHA, and it remains to be determined how these will compete with, or complement each other.

On the theoretical side, our strategy is to iterate on the software design in order to gradually make it more crisp, and thus to move in the direction of an implementation that can be validated formally.

The DORP software is a separately maintained system variant where deliberation and agenda management facilities are checked out with an alternative set of actions. This testbench is particularly useful for verifying the consistency of the timing when actions are invoked, scripts are executed, and events and activities are recognized along a common timeline. Facilities that have been validated in DORP as working correctly are routinely transferred to DOSAR.

Both DOSAR and DORP have been implemented in CommonLisp in order to prepare the ground for the theoretical arm of the work. They are running in standard PC environments. Achieving the required real-time performance has never been a limiting factor in this work. The Software Individuals Architecture [21] is used as the lowest Lisp software layer, providing a systematic way of structuring software modules, data directories, command sessions, and software maintenance and exchange.

10 Conclusion

We have described the mutual relation between the Double Helix Architecture and the Cognitive Robotics Logic. Our main message is that this agent architecture, which explicitly describes the location of different computational processes and layers along a time axis with discrete steps, is useful for clarifying both the organization of the computation, the semantics of the logic, and the interdependence between those two issues.

Acknowledgments

The WIT'S project is funded by a generous grant from the Cnut and Alice Wallenberg Foundation.

References

1. Raja Chatila and Jean-Paul Laumond. Position referencing and consistent world modelling on mobile robots. In *IEEE International Conference of Robotics and Automation*, pages 138–145, 1985.
2. Thomas Dean and Michael P. Wellman. *Planning and Control.* Morgan-Kaufmann, 1991.
3. Patrick Doherty, Gösta Granlund, Krzystof Kuchcinski, Erik Sandewall, Klas Nordberg, Erik Skarman, and Johan Wiklund. The witas unmanned aerial vehicle project. In *European Conference on Artificial Intelligence*, pages 747–755, 2000.
4. Patrick Doherty, Joakim Gustafsson, Lars Karlsson, and Jonas Kvarnström. Temporal action logics language. specification and tutorial. *Electronic Transactions on Artificial Intelligence*, 2:273–306, 1998.
5. Patrick Doherty, Tommy Persson, Björn Wingman, Patrick Haslum, and Fredrik Heintz. A CORBA-based deliberative/reactive architecture for unmanned aerial vehicles. Unpublished manuscript, 2002.
6. R. James Firby, Roger E. Kahn, Peter N. Prokopowicz, and Michael J. Swain. An architecture for vision and action. In *International Joint Conference on Artificial Intelligence*, pages 72–79, 1995.
7. Oliver Lemon, Alexander Gruenstein, and Stanley Peters. Collaborative activities and multi-tasking in dialogue systems. *Traitement Automatique des Langues, special issue on dialogue*, 2002. To appear.
8. Hector Levesque, Fiora Pirri, and Ray Reiter. Foundations for the situation calculus. *Electronic Transactions on Artificial Intelligence*, 2:159–178, 1998.
9. Hector J. Levesque, Raymond Reiter, Yves Lesérance, Fangzhen Lin, and Richard B. Scherl. Golog: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31(1-3):59–84, April-June 1997.
10. Jörg Müller. Control architectures for autonomous and interacting agents. A survey. In Lawrence Cavedon, Anand Rao, and Wayne Wobcke, editors, *Intelligent Agent Systems*, pages 1–26, 1996.
11. Jörg Müller. *The Design of Intelligent Agents.* Springer Verlag, 1996.
12. Nils Nilsson. Teleo-reactive programs and the triple tower architecture. *Electronic Transactions on Artificial Intelligence*, 5B:99–110, 2001.
13. Anand S. Rao and Michael P. Georgeff. An abstract architecture for rational agents. In *International Conference on Knowledge Representation and Reasoning*, pages 439–449, 1992.
14. Stuart Russell and Peter Norvig. *Artificial Intelligence. A Modern Approach.* Prentice-Hall, 1995.
15. Erik Sandewall. Combining logic and differential equations for describing real-world systems. In *Proc. International Conference on Knowledge Representation, Toronto, Canada*, 1989.
16. Erik Sandewall. Filter preferential entailment for the logic of action in almost continuous worlds. In *International Joint Conference on Artificial Intelligence*, pages 894–899, 1989.
17. Erik Sandewall. *Features and Fluents. The Representation of Knowledge about Dynamical Systems. Volume I.* Oxford University Press, 1994.
18. Erik Sandewall. Towards the validation of high-level action descriptions from their low-level definitions. *Artificial Intelligence Communications*, December 1996. Also Linköping University Electronic Press, `http://www.ep.liu.se/cis/1996/004/`.

19. Erik Sandewall. A logic-based characterization of goal-directed behavior. *Electronic Transactions on Artificial Intelligence*, 1:105–128, 1997.
20. Erik Sandewall. Cognitive robotics logic and its metatheory: Features and fluents revisited. *Electronic Transactions on Artificial Intelligence*, 2:307–329, 1998.
21. Erik Sandewall. On the design of software individuals. *Electronic Transactions on Artificial Intelligence*, 5B:143–160, 2001.
22. George N. Saridis and Kimon P. Valavanis. On the theory of intelligent controls. In *Proc. of the SPIE Conference on Advances in Intelligent Robotic Systems*, pages 488–495, 1987.
23. Murray Shanahan. A logical account of the common sense informatic situation for a mobile robot. *Electronic Transactions on Artificial Intelligence*, 2:69–104, 1998.
24. D. van Dalen. *Logic and Structure*. Springer Verlag, 1980.

The DD&P Robot Control Architecture*
A Preliminary Report

Frank Schönherr and Joachim Hertzberg

Fraunhofer - Institute for Autonomous intelligent Systems (AIS)
Schloss Birlinghoven, 53754 Sankt Augustin, Germany
{schoenherr,hertzberg}@ais.fraunhofer.de

Abstract. The paper presents current results of our work on DD&P, a two layer control architecture for autonomous robots. DD&P comprises a deliberative and a behavior-based part as two peer modules with no hierarchy among these two layers, as sketched in [HJZM98]. Interaction between these control layers is regulated by the *structure* of the information flow, extending the "classical" sense-model-plan-act principle. The paper stresses two architectural highlights of our approach: The implementation of the "Plan-as-Advice" principle to execute plan operators by the behavior-based part, and the grounding of the symbolic planner description via chronicle recognition.

1 Background and Overview

There are several good reasons to include a behavior-based component in the control of an autonomous mobile robot. There are equally good reasons to include in addition a deliberative component. Having components of both types results in a *hybrid* control architecture, intertwining the behavior-based and the deliberative processes that go on in parallel. Together, they allow the robot to react to the dynamics and unpredictability of its environment without forgetting the high-level goals to accomplish. Arkin [Ark98, Ch. 6] presents a detailed argument and surveys hybrid control architectures; the many working autonomous robots that use hybrid architectures include the Remote Agent Project [MNPW98,Rem00] as their highest-flying example.

While hybrid, layered control architectures for autonomous robots, such as Saphira [KMSR97] or 3T [BFG+97] are state of the art, some problems remain that make it a still complicated task to build a control system for a concrete robot to work on a concrete task. To quote Arkin [Ark98, p. 207],

> the nature of the boundary between deliberation and reactive execution is not well understood at this time, leading to somewhat arbitrary architectural decisions.

* This work is partially supported by the German Fed. Ministry for Education and Research (BMBF) in the joint project AgenTec (01AK905B), see http://www.agentec.de for details.

M. Beetz et al. (Eds.): Plan-Based Control of Robotic Agents, LNAI 2466, pp. 249–269, 2002.

This boundary can be split into two different sub-problems:

(1) The symbolic world representation *update* problem, which is an instance of the *symbol grounding problem* [Har90]. There are solutions to important parts of that problem, such as methods and algorithms for sensor-based localization to reason about future navigation actions: [FBT99] presents one of the many examples for on-line robot pose determination based on laser scans. If the purpose of deliberation is supposed to be more general than navigation, such as action planning or reasoning about action, then the need arises to sense more generally the recent relevant part of the world state and update its symbolic representation based on these sensor data. We call this representation the current *situation.*

The naive version of the update problem "Tell me all that is currently true about the world!" needs not be solved, luckily, if the goal is to build a concrete robot to work on a concrete task. *Only those facts need updating that, according to the symbolic domain model used for deliberation, are relevant for the robot to work on its task.* Then, every robot has its *sensor horizon,* i.e., a border in space and time limiting its sensor range. The term sensor is understood in a broad sense: It includes technical sensors like laser scanners, ultra sound transducers, or cameras; but if, for example, the arena of a delivery robot includes access to the control of an elevator, then a status request by wireless Ethernet to determine the current location of the elevator cabin is a sensor action, and the elevator status is permanently within the sensor horizon. This information is completed by invariable world state descriptions e.g., global maps of the environment. We assume: *The persistent world state together with the variable information within the sensor horizon is sufficient to achieve satisfying robot performance.*

This said, the task of keeping the facts of a situation up-to-date remains to continually compute from recent sensor data and the previous situation a new version of the situation as far as it lies within the sensor horizon. The computation is based on plain, current sensor values as well as histories of situations and sensor readings or aggregates thereof. Practically, we cannot expect to get accurate situation updates instantly; all we can do is make the situation update as recent, comprehensive, and accurate as possible.

(2) The *execution* of symbolic plans in the physical world. Given a plan, the robot must, first, determine whether the plan is still valid and which step is next for execution, and, second, act in the world according to this current plan step. At present DD&P offers nothing original for the first part, and this paper does not go into any detail concerning it. As in many other hybrid robot control systems in the sequence of STRIPS/Shakey's triangle tables [FHN72], we assume that an execution monitor component permanently determines the progress and feasibility of the recent plan, based on the permanently ongoing symbolic world representation update just described.

For the part of acting concretely according to the current symbolic plan step, the problem is that the abstraction from worldly detail that is wanted and unavoidable for effective planning, makes the translation into, say, physical motion non-unique. On the one hand, immediate reaction to unforeseen events, such as navigating around persons while following a planned trajectory in a

crowded hallway, has to dominate the to-be-executed plan: reaction overwrites plan step execution. On the other hand, the robot must stick to its plan, tolerating momentary disadvantages over greedy reactive behavior, to achieve coherent, goal-oriented performance in the long run: plan step execution overwrites reaction. So, executing a plan means permanently negotiating between the forces urging to react and the desire to stick with the plan. Every hybrid robot control architecture must cope with that.

Our view of building hybrid robot controllers involving a behavior-based robot control system (BBS)[Mat99] as reactive component is shaped by our work in progress on the DD&P robot control architecture [HJZM98,HS01,SCHC01]. The demo example we will present is formulated in a concrete BBS framework, namely, Dual Dynamics (DD, [JC97]) and we will illustrate how to blend it with classical propositional action planners.

The rest of this paper is organized as follows. In Sec. 2, we present our approach of formulating BBSs as dynamical systems and give a detailed example in Sec. 3. Sec. 4 discusses our view on a concrete planning system in general. Sec. 5 and Sec. 6 contain the technical contribution of the paper: we first sketch how we assume the deliberation component interferes the BBS control component, and then describe the technique of extracting facts from BBS activation value histories. Sec. 7 discusses the approach and relates it to the literature. Sec. 8 concludes.

2 BBSs as Dynamical Systems

We assume a BBS consists of two kinds of behaviors: low-level behaviors (LLBs), which are directly connected to the robot actuators, and higher-level behaviors (HLBs), which are connected to LLBs and/or HLBs. Each LLB implements two distinct functions: a target function and an activation function. The target function for the behavior b provides the reference t_b for the robot actuators (*"what to do"*) as follows:

$$t_b = f_b(s^T, s_f^T, \alpha_{LLB}^T) \tag{1}$$

where f_b is a nonlinear vector function with one component for each actuator variable, s^T is the vector of all inputs from sensors, s_f^T is the vector of the sensor-filters and α_{LLB}^T is the vector of activation values of the LLBs. By sensor-filters – sometimes called virtual sensors – we mean *Markovian* and *non-Markovian* functions used for processing specific information from sensors.

The LLB activation function *modulates* the output of the target function. It provides a value between 1 and 0, meaning that the behavior fully influences, does not influence or influences to some degree the robot actuators. It describes *when* to activate a behavior. For LLB b the activation value is computed from the following *differential equation*:

$$\dot{\alpha}_{b,LLB} = g_b(\alpha_{b,LLB}, OnF_b, OffF_b, OCT_b) \tag{2}$$

Equation 2 gives the variation of the *activation value* $\alpha_{b,LLB}$ of this LLB. g_b is a nonlinear function. OCT_b allows the planner to influence the activation values, see Sec. 5. The scalar variables OnF_b and $OffF_b$ are computed as follows:

$$OnF_b = u_b(s^T, s_f^T, \alpha_{LLB}^T, \alpha_{HLB}^T) \tag{3}$$

$$OffF_b = v_b(s^T, s_f^T, \alpha_{LLB}^T, \alpha_{HLB}^T) \tag{4}$$

where u_b and v_b are nonlinear functions. The variable OnF_b sums up all conditions which recommend activating the respective behavior (on forces) and $OffF_b$ stands for adversary conditions to the respective behavior (off forces). The definitions in Eqs. 2 - 4 are exemplified in the next section.

The HLBs implement only the activation function. They are allowed to modulate only the LLBs or other HLBs on the same or *lower* level. In our case, the change of activation values for the HLBs $\alpha_{b,HLB}$ are computed in the same manner as Eq. 2. To result in a stable control system, the levels must "run" on a different *timescale*; HLBs change activation only on a longer term, LLBs on a shorter term.

In the original Dual Dynamics paper [JC97] Eq. 1 was also implemented as a differential equation – therefore the name Dual Dynamics. The global *motor controller* is now integrated into the microcontroller of our robot, making behavior design independent from effects like battery values or floor surface. This avoids the integration of the *closed-loop* controller locally into *every* target function and allows the use of much easier to handle absolute output values. The overall result is very similar to the original approach.

The reason for updating behavior activation in the form of Eq. 2 is this. By referring to the previous activation value α_b, it incorporates a memory of the previous evolution which can be overwritten in case of sudden and relevant changes in the environment, but normally prevents activation values from exhibiting high-frequency oscillations or spikes. At the same time, this form of the activation function provides some *low-pass filtering* capabilities, damping sensor noise or oscillating sensor readings.

Independent from that, it helps to develop stable robot controllers if behavior activations have a tendency of moving towards their boundary values, i.e., 0 or 1 in our formulation. To achieve that, we have implemented g_b in Eq. 2 as a *bistable* ground form (see [BK01] Sec. 4 for details) providing some *hysteresis effect*. Without further influence, this function pushes activation values lower/higher than some threshold β (typically $\beta = 0.5$) softly to 0/1. The activation value changes as a result of *adding* the effects coming from the variables OnF, $OffF$, OCT and the bistable ground form. Exact formulations of the g_b function are then just technical and unimportant for this paper, [KJ02] gives a detailed description.

The relative smoothness of activation values achieved by using differential equations and bistability will be helpful later in the technical contribution of this paper , when it comes to derive facts from the time series of activation values of the behaviors (Sec. 6).

In our BBS formulation, behavior arbitration is achieved using the activation values. As shown in Eqs. 2 - 4, each behavior can interact with (i.e., encourage or inhibit) every other behavior on the same or lower level. The model of interaction between behaviors is defined by the variables OnF and $OffF$.

The output vector or *reference vector* r of the BBS for the robot actuators is generated by summing all LLB outputs by a *mixer*, as follows:

$$r = \kappa \sum_b \alpha_{b,LLB} t_b \tag{5}$$

where $\kappa = (1/\sum_b \alpha_{b,LLB})$ is a normalization factor.

Together with the form of the activation values, this way of blending the outputs of LLBs avoids *discontinuities* in the reference values r_i for the single robots actuators, such as sudden changes from full speed forward to full speed backward. One could even use voting, "winner-takes-all" or priority-based behavior selection mechanisms very easily, if desired.

DD has recently been extended with *team variables* [BK01], which are exported by a behavior system, so that other behavior systems (robots) can read their current values. For every team variable only the exporting behavior systems is allowed to change its value, for all others it's value is read-only. This mechanism adds multi-robot cooperation capabilities into the DD framework, fitting perfectly in the global context of agent technology [Age] in which we are doing this work.

DD differs from most other BBS, e.g. [Ark98,SKR95,Ros97], by using dynamical system theory for the definition and *analysis* of behaviors. Furthermore this structure supports a practical interface for the integration of a deliberative control layer, see Sec. 5, 6. The implementation, evaluation and analysis of DD behaviors for different types of robots is supported by a set of tools [Bre00,BGG+99], cf. Fig.1.

3 An Example

To illustrate the notation of Sec. 2 we give a demonstration problem consisting of the task of following a wall with a robot and entering only those doors that are wide enough to allow the entrance. Figure 2 gives an overview about the main part of our arena. The depicted robot is equipped with a short distance laser-scanner, 4 infrared side-sensors, 4 front/back bumpers and some dead-reckoning capabilities.

We used a simulator based on the DDDesigner prototype tool [Bre00], [BGG+99]. The tool allows checking isolated behaviors or the whole BBS in designated environmental situations (configurations). The control system contains three HLBs and six LLBs, depicted in Tab. 1, with $RobotDirection$ and $RobotVelocity$ as references for the two respective actuators.

Most of the implemented behaviors are common for this kind of tasks. However, we decided to split the task of passing a door in a sequence of two LLBs. This helps structure, maintain and independently improve these two behaviors.

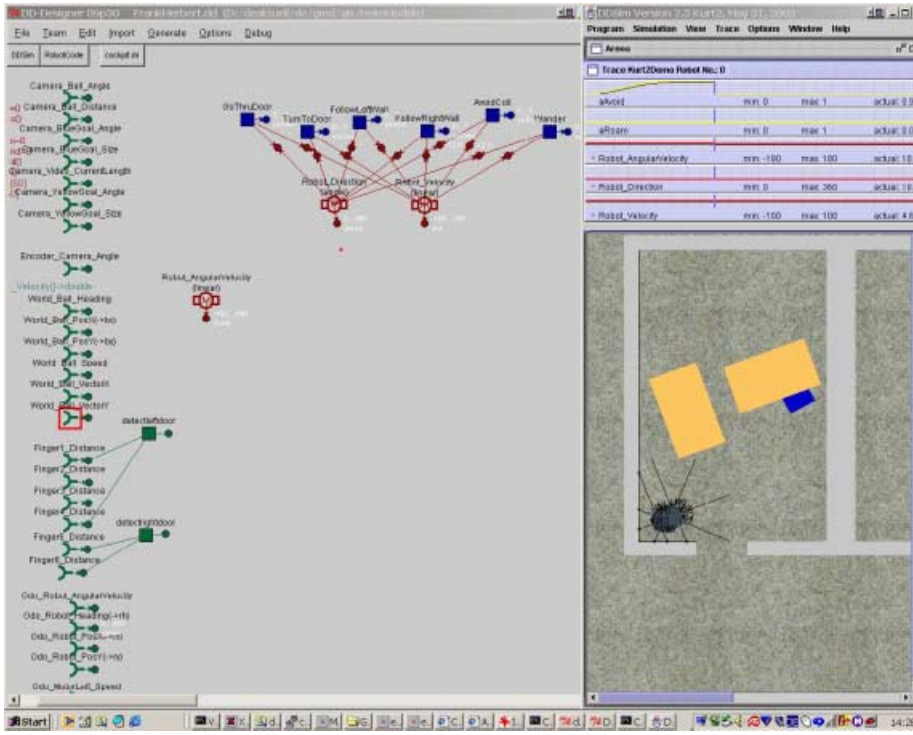


Fig. 1. Screen shots from the most recent version of the DDDesigner tools [Bre00,BGG+99]. These tools mainly comprise an graphical user interface, where DD models can be specified in an intuitive high-level language; automatic code generators for simulation, monitoring, and different kinds of robot hardware; a simulator; and a monitoring environment for wireless online tracing of information processing on board the running robot

The implementation of these behaviors used in this study are deliberately non-sophisticated in order to test the power of the chronicle recognition described below (Sec. 6) in the presence of unreliable behaviors.

To give a simplified example of BBS modeling, here are the "internals" of Wander:

$$OnF_{\mathsf{Wander}} = k_1(1 - \alpha_{\mathsf{CloseToDoor}}) * (1 - \alpha_{\mathsf{FollowRightWall}}) * (1 - \alpha_{\mathsf{FollowLeftWall}}) * (1 - \alpha_{\mathsf{AvoidColl}}) \quad (6)$$

$$OffF_{\mathsf{Wander}} = k_2\alpha_{\mathsf{AvoidColl}} + k_3\alpha_{\mathsf{CloseToDoor}} + k_4\alpha_{\mathsf{FollowRightWall}} + k_5\alpha_{\mathsf{FollowLeftWall}} \quad (7)$$

$$RobotDirection_{\mathsf{Wander}} = randomDirection() \quad (8)$$

$$RobotVelocity_{\mathsf{Wander}} = mediumSpeed \quad (9)$$

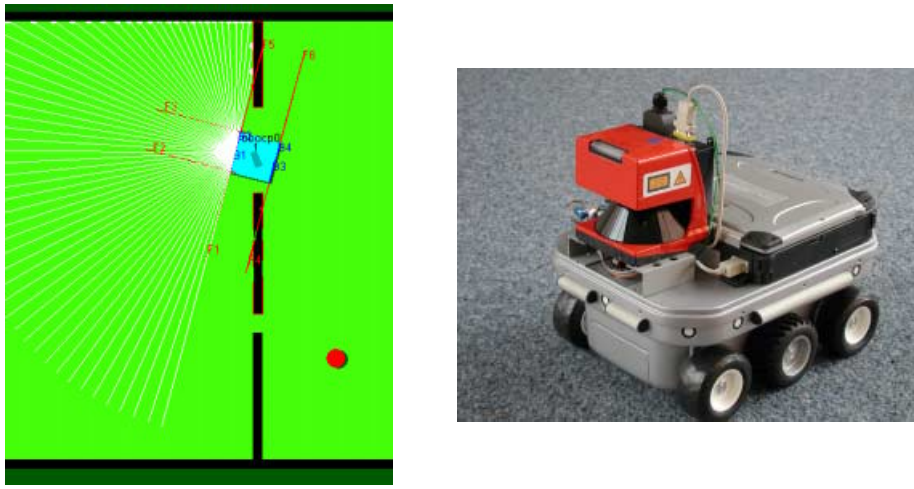


Fig. 2. The left picture shows the demo arena and the final robot pose in Example 1. The robot has started its course at the lower right corner, below to the round obstacle. The right picture shows our office navigation robot KURT2 which was simulated in the example

$$t_{\mathsf{Wander}} = \begin{bmatrix} RobotDirection_{\mathsf{Wander}} \\ RobotVelocity_{\mathsf{Wander}} \end{bmatrix} \tag{10}$$

$$\dot{\alpha}_{\mathsf{Wander}} = g_{\mathsf{Wander}}(\alpha_{\mathsf{Wander}}, OnF_{\mathsf{Wdr}}, OffF_{\mathsf{Wdr}}, OCT_{\mathsf{Wdr}}) \tag{11}$$

where $k_1 \ldots k_5$ are empirically chosen constants. $randomDirection()$ could be every function that generates a direction which results in a randomly chosen trajectory.

Due to its *product* form, OnF_{Wander} can only be significantly greater than zero if all included α_b are approximately zero. $OffF_{\mathsf{Wander}}$ consists of a *sum* of terms allowing every included behavior to deactivate Wander. Both terms are simple and can be calculated extremely fast, which is a guideline for most BBSs. The OCT term will be explained in the next section. In general, DD terms have a very modular structure – sum of products – which permits thoughtful adaptability to new situations and/or behaviors.

Fig. 3 shows the *activation value histories* generated during a robot run, which will be referred to as Example 1. The robot starts at the right lower edge of the arena in Fig. 2 with Wander in control for a very short time, until a wall is perceived. This effect is explained by Eq. 6. While the robot starts to follow the wall, it detects the small round obstacle in front. In consequence, two LLBs are active simultaneously: AvoidColl and FollowLeftWall. Finally, the robot follows the wall, ignores the little gap and enters the door. In the examples for this paper, FollowRightWall is always inactive and therefore not shown in the activation value curves.

This exemplifies the purpose of a *slow* increase in behavior activation. FollowLeftWall should only have a strong influence to the overall robot behavior

Table 1. Overview of the Example 1 behaviors. Even if some of their names e.g., CloseToDoor and InCorridor, suggest predicate names, they are all *behaviors*

The HLBs are the following:	
CloseToDoor	is activated if there is evidence for a door;
InCorridor	is active while the robot moves inside a corridor;
TimeOut	was implemented in order to avoid getting stuck in a situation, like two robots blocking each other by trying to pass a door from the opposite sides;
The LLBs are the following:	
TurnToDoor	is depending on CloseToDoor and gets activated if the robot is situated on a level with a door;
GoThruDoor	is activated after the behavior TurnToDoor was successful;
FollowRightWall	is active when a right wall is followed;
FollowLeftWall	is active when a left wall is followed;
AvoidColl	is active when there is an obstacle in the front of the robot
Wander	is active when no other LLB is active.

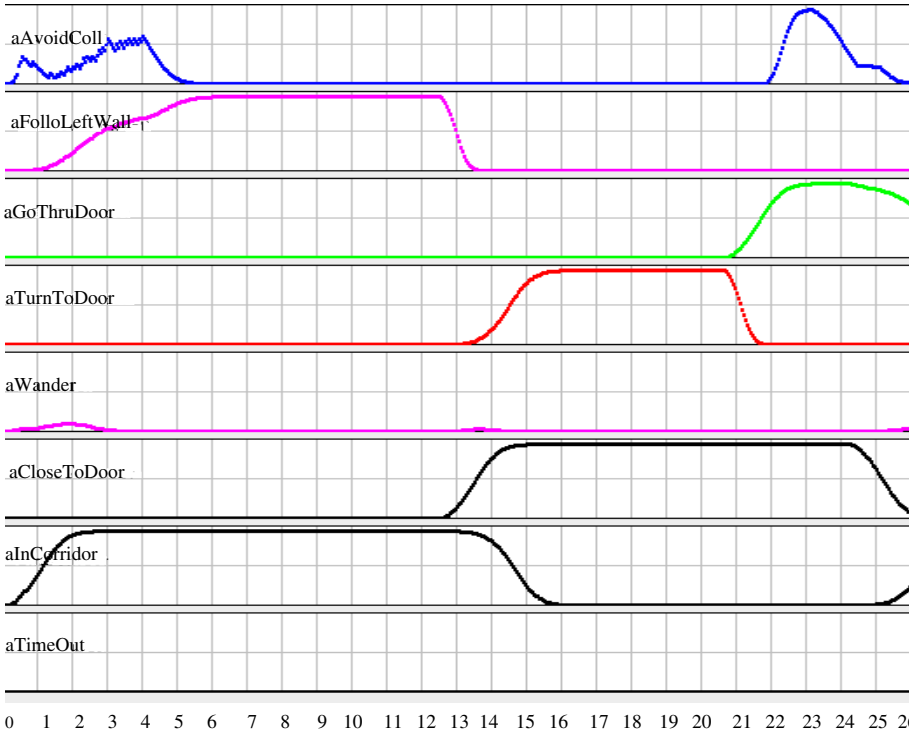


Fig. 3. The activation value histories for Example 1. The numbers are time unit for reference

if a wall is perceived with *both* side-sensors for some time, so as to be more sure that the robot really has sensed a wall. The small dent in the activation of FollowLeftWall (around the time $t = 4$) is explained by perceiving free space with one side-sensor. If both side-sensors detect free space this behavior would be deactivated. The turning to the door is described by rising/falling edges of some activation values. The second rise of AvoidColl (after $t = 22$) is caused by the door frame, which pops into sight as a close obstacle at the very end of the turning maneuver. Effectively, the collision avoidance guides the robot through the door. Finally GoThruDoor gets slowly deactivated allowing other behaviors to take control of the robot.

The HLBs CloseToDoor and InCorridor describe global states, thereby modulating the interaction, activation and sequencing of the LLBs. Some other remarks on behavior arbitration in DD: Our approach is neither purely competitive nor cooperative, allowing internal or external constraints and degree of freedom to be reflected flexibly. For example TurnToDoor and FollowLeftWall are mutually exclusive, while FollowLeftWall and AvoidColl are cooperative.

4 Deliberation in DD&P

At the current work state we have identified some constraints on useful planning systems which arise from the targeted application area [Age], which comprises industrial production, delivery services or inaccessible environments. *Soundness* and *completeness* of the planning system are non-relaxable restrictions in service robotics. Otherwise we could get no guarantee that plans for solvable missions get in fact generated, which is requested by most service robotic clients. The usage of *heuristics* has to be considered very carefully under this assumption. From the conceptual point of view, DD&P is able to work with almost any kind of planner, which we will show in the next section. Please not that the successful plan generation means in now way the successful plan execution. We just want to exclude additional sources of uncertainty from the planning component.

One should be aware that even a reliable knowledge base update process is not immune against generating *irrelevant* facts. Irrelevant means that these facts are not needed for any solution of the current planning problem. They can enlarge the planner's search space and thereby its search time dramatically. There is work ongoing about solving this problem from the planner side, like e.g. RIFO [NDK97] – which has the disadvantage of not being completely solution preserving.

We are not developing our own planner(s). We are modeling our symbolic task description in (subsets of) PDDL 2.1 [FL01] which enables us to use the full variety of "off-the-shelf planners" that can be downloaded from the Internet. Our idea is to use existing and fast *propositional* systems which generate even non trivial plans very quickly. The hypothesis is that a "throw-away-plan" approach is practically superior to the use of more sophisticated planners. A probabilistic one promises more reliable plans, yet taking much more time for generation. Furthermore assigning probabilities is far from trivial.

In terms of implementation (see Fig. 4), we are currently thinking about IPP [KNHD97] as it adds to pure propositional planning some expressivity (ADL operators) which can be easily expressed in PDDL. An HTN-Planner like SHOP [NCLMA99] is a potential alternative. It permits the use of domain knowledge by decomposing tasks into subtasks and handling domain constraints. This even includes a kind of *plan correction* capability by backtracking other decompositions of high-level (sub-)goals in case of unexecutable actions.

Note that we are not restricting the deliberation part to exactly one planner or to one single planning level. Plans may make sense for a robot in different varieties, on different, possibly unconnected levels of abstraction, and on different degrees of commitment. For example, a path plan and an action plan differ in specificity and, consequently, can efficiently be generated by different algorithms, yet they are both part of the robot's deliberation. Plans on a social level tend to be more abstract than those on individual robot mission level. To yield a satisfying and coherent overall performance, such different plans need to be coordinated in the end. However, DD&P does not achieve this harmonization by forcing the robot domain modeler to employ exactly one planner, but currently leaves open the structure of the deliberative control part.

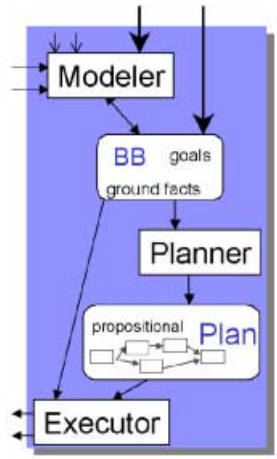


Fig. 4. A "closer-view" on the deliberation part of DD&P. As we do not assume that the system's knowledge base is accurate, it is more aptly called belief base (BB). It includes the current world state and goals. The *Modeler* is updating the world state in the BB, via chronicle recognition (see Sec. 6) or external input, e.g. form other agents. The *Executor* is biasing behaviors towards the current to be executed action, and monitoring its results. See text for details

From the propositional planner point of view execution of an action in the current plan is finished if its specified postconditions hold in the knowledge base or if a replanning process is started. The first one permits skipping of unnecessary

actions like opening an already open door. These kind of symbolic actions can arise from different conditions, e.g. imperfect knowledge base entries. Replanning can occur if a *time-out condition* signals a flaw in the execution, the final state of the current plan is achieved, or a serious change in the knowledge-base has been recognized. Serious changes means a new "high-priority" goal or dramatic changes to the current plan, like an inoperative elevator which "cuts off" wide operation areas from the robot. How to define and detect such serious changes is a non-trivial task and lies out of the scope of this paper.

5 From Symbols to Action: Blending Behaviors with Operators

This section sketches the control flow from the planner to the BBS. The basic idea is this: One of the action planners discussed in Sec. 4 continually maintains a current action plan, based on the current situation and the current set of externally-provided or self-generated mission goals. Based on the current plan and the current situation, an execution component picks one of the operators in the plan as the one currently to be executed. Plan execution is done in a *plans-as-advice* [Pol92,PRK90,Suc87] fashion: Executing an operator means stimulating more or less strongly the behaviors working in favor of the operator, and muting those working against its purpose. This does not mean rigid control, rather giving some degree of freedom to the BBS to react to unforeseen situations, like obstacles, batteries going down or other events which we discuss at the end of Sec. 7, basing concrete execution in the world more on fresh sensor data than on the planner's advice. Which operator stimulates or mutes which behaviors is an information that the domain modeler has to provide along with the domain model for the deliberative component and the set of behaviors for the BBS.

Therefore we first introduce the so-called *influence-matrix* $\mathbf{I}$ with $I_{b,op} \in \mathbb{R}$, which encodes whether and how the current ground operator $op \in OP$ influences the behavior b through the term OCT_b.

Technically, the influence of the *current* operator is "injected" into the BBS in terms of the *Operator-Coupling-Terms* (OCT) in the activation functions, see Eqs. 2. The influence of the *current* ground operator *op* gets inside every behavior b through the term OCT_b, as follows:

$$OCT_b = |I_{b,op}|(sign^+(I_{b,op}) - \alpha_b) \tag{12}$$

where $sign^+$ is $+1$ if $x > 0$ and 0 else. $I_{b,op} \not= 0$ iff *op* influences the behavior b. Its sign expresses whether the operator influence is of the stimulating or muting sort: If $I_{b,op} > 0$, then the respective behavior is stimulated, and muted if $I_{b,op} < 0$. The absolute value of $I_{b,op}$ models the strength of the operator influence on the behavior b.

To give an example, assume that the domain model for the deliberation component includes an operator variable **GO-IN-RM**(x) modeling the action of some office delivery robot to go (from wherever it is) to and enter room x.

Let the behavior inventory be the one specified in Sec. 3. Here is a selection of $I_{b,\textbf{GO-IN-RM}(x)}$-column for the influence of a ground operator like **GO-IN-RM**(A) for a concrete room A:

Table 2. A selection of the influence-matrix I. See text for details

	. . . .	GO-IN-RM(x)	. . . .
CloseToDoor	. . . .	5	. . . .
InCorridor	. . . .	0	. . . .
TimeOut	. . . .	0	. . . .
TurnToDoor	. . . .	0	. . . .
GoThruDoor	. . . .	0	. . . .
FollowRightWall	. . . .	-5	. . . .
FollowLeftWall	. . . .	-5	. . . .
AvoidColl	. . . .	0	. . . .
Wander	. . . .	-2	. . . .

Note that in Tab. 2 $I_{b,op} \in [-40, 40]$ which is in relation to the actual implementation of low-level behaviors "running" with 40 iterations per second. To discuss some values in detail: $I_{\text{CloseToDoor},\textbf{GO-IN-RM}(x)} \not= 0$ means that this behavior is influenced if **GO-IN-RM**(x) is chosen to be executed. In this case, its large *positive* value stimulates CloseToDoor strongly. With the same strength FollowRightWall and FollowLeftWall are muted while Wander is only muted slightly.

The zero influence on TurnToDoor or GoThruDoor does in no way mean that it is not needed to execute **GO-IN-RM**(x). The activation of TurnToDoor is directly depending on the sensor context – perceiving a gap – and on the activation of CloseToDoor which is influenced by this operator. Furthermore GoThruDoor can only be activated in a successful sequence with TurnToDoor, see Sec. 3.

On the other side **GO-IN-RM**(x) can only be chosen for execution if its precondition CloseTo(x) is true in the knowledge base, which also gives strong evidence to be close to the door of room A. Accordingly we are combining the already available capabilities of the BBS in a natural way with that one of the deliberative part.

6 From (Re-)Action to Symbols: Extracting Facts from Activation Values

We now turn to the method for extracting facts from activation value histories. It is influenced by previous work on chronicle recognition, such as [Gha96].

To start, take another look at the activation curves in Fig. 3 in Sec. 3. Some irregular activation time series occur due to the dynamics of the robot/environment interaction, such as early in the AvoidColl and Wander behaviors. However, certain patterns re-occur for single behaviors within intervals of time, such as a

value being more or less constantly high or low, and values going up from low to high or vice versa. The idea to extract symbolic facts from activation values is to consider characteristic groups or *gestalts* of such qualitative activation features occurring in *chronicles* over time.

To make this precise, we define, first, *qualitative activation values* (or briefly, qualitative activations) describing these isolated patterns. In this paper, we consider four of them, which are sufficient for defining and demonstrating the principle, namely, rising/falling edge, high and low, symbolized by predicates $\Uparrow\mathsf{e}$, $\Downarrow\mathsf{e}$, Hi, and Lo, respectively. In general, there may be more qualitative activations of interest, such as a value staying in a medium range over some period of time. For a behavior b and time interval $[t_1, t_2]$, they are defined as

$$\begin{aligned}
\mathsf{Hi}(b)[t_1,t_2] &\equiv \alpha_b[t] \geq h \text{ for all } t_1 \leq t \leq t_2 \\
\mathsf{Lo}(b)[t_1,t_2] &\equiv \alpha_b[t] \leq l \text{ for all } t_1 \leq t \leq t_2 \\
\Uparrow\mathsf{e}(b)[t_1,t_2] &\equiv \alpha_b[t_1] = l \text{ and } \alpha_b[t_2] = h \text{ and} \qquad (13)\\
&\quad\ \alpha_b \text{ increases } \textit{generally monotonically} \text{ over } [t_1,t_2] \\
\Downarrow\mathsf{e}(b)[t_1,t_2] &\equiv \alpha_b[t_1] = h \text{ and } \alpha_b[t_2] = l \text{ and} \qquad (14)\\
&\quad\ \alpha_b \text{ decreases } \textit{generally monotonically} \text{ over } [t_1,t_2]
\end{aligned}$$

for given threshold values $0 \ll h \leq 1$ and $0 \leq l \ll 1$, where $\alpha_b[t]$ denotes the value of α_b at time t. *General monotonicity* requires another technical definition. We are using a simple first order discrete time filter, the details of which are beyond the scope of this paper. The idea is that some degree of noise should be allowed in, e.g., an increasing edge, making the increase locally non-monotonic. In the rather benign example activation curves in this paper, regular monotonicity suffices. Similarly, it is not always reasonable to use the global constants h, l as Hi and Lo thresholds, respectively. It is possible to use different threshold constants or thresholding functions for different behaviors. We do not go into that here. Then, it makes sense to require a minimum duration for $[t_1, t_2]$ to prevent useless mini intervals of Hi and Lo types from being identified. Finally, the strict equalities in Eqs. 13 and 14 are unrealistic in real robot applications, where two real numbers must be compared, which are seldom strictly equal. Equality $\pm\epsilon$ is the solution of choice here.

The key idea to extract facts from activation histories is to consider patterns of qualitative activations of several behaviors that occur within the same interval of time. We call these patterns *activation gestalts.* We express them formally by a time-dependent predicate AG over a set Q of qualitative activations of potentially many behaviors. For a time interval $[t, t']$ the truth of $AG(Q)[t, t']$ is defined as the conjunction of conditions on the component qualitative activations $q \in Q$ of behaviors b in the following way:

$$\begin{aligned}
&\text{case } q = \mathsf{Hi}(b) \text{ then } \mathsf{Hi}(b)[t,t'] \\
&\text{case } q = \mathsf{Lo}(b) \text{ then } \mathsf{Lo}(b)[t,t'] \\
&\text{case } q = \Uparrow\mathsf{e}(b) \text{ then } \Uparrow\mathsf{e}(b)[t_1,t_2] \text{ for some } [t_1,t_2] \subseteq [t,t'], \\
&\qquad\qquad\qquad\qquad \text{and } \mathsf{Lo}(b)[t,t_1] \text{ and } \mathsf{Hi}(b)[t_2,t']
\end{aligned}$$

$$\text{case } q = \Downarrow\mathsf{e}(b) \text{ then } \Downarrow\mathsf{e}(b)[t_1, t_2] \text{ for some } [t_1, t_2] \subseteq [t, t'], \\ \text{and } \mathsf{Hi}(b)[t, t_1] \text{ and } \mathsf{Lo}(b)[t_2, t']$$

Note that it is not required that different rising or falling edges in Q start or end synchronously among each other or at the interval borders of $[t, t']$—they only must all occur somewhere within that interval.

For example, $AG(\{\Uparrow\mathsf{e}(\mathsf{GoThruDoor}), \Downarrow\mathsf{e}(\mathsf{TurnToDoor}), \mathsf{Hi}(\mathsf{CloseToDoor})\})$ is true over $[20, 24]$ in the activation histories in Fig. 3; it is also true over $[16, 23]$ (and therefore, also over their union $[16, 24]$), but not over $[16, 25]$, as CloseToDoor has left its Hi band by time 25, and possibly the same for GoThruDoor, depending on the concrete value of the h threshold.

A *chronicle* over some interval of time $[t_0, t]$ is a set of activation gestalts over sub-intervals of $[t_0, t]$ with a finite set of n linearly ordered internal *interval boundary points* $t_0 < t_1 < \cdots < t_n < t$. A ground fact is extracted from the activation history of a BBS as true (or rather, as *evident*, see the discussion below) at time t if its *defining chronicle* has been observed over some interval of time ending at t. The defining chronicle must be provided by the domain modeler, of course.

We give as an example the defining chronicle of the fact InRoom that the robot is in some room, such as the one left of the wall in Fig. 2. InRoom$[t]$ is extracted if the following defining chronicle (15) is true within the interval $[t_0, t]$, where the t_i are existentially quantified. Combining a complex sequence of conditions takes into account the wilful simplicity and unreliability of our BBS (see Sec. 3). Accordingly, the resulting – *reliable* – chronicle looks a little bit longish:

$$\begin{aligned} & AG(\{\Downarrow\mathsf{e}(\mathsf{GoThruDoor})\})[t_4, t] \\ \wedge\ & AG(\{\Uparrow\mathsf{e}(\mathsf{GoThruDoor}), \Downarrow\mathsf{e}(\mathsf{TurnToDoor}), \mathsf{Hi}(\mathsf{CloseToDoor})\})[t_3, t_4] \\ \wedge\ & AG(\{\mathsf{Hi}(\mathsf{TurnToDoor}, \mathsf{Lo}(\mathsf{InCorridor})\})[t_2, t_3] \\ \wedge\ & AG(\{\Uparrow\mathsf{e}(\mathsf{TurnToDoor}), \Uparrow\mathsf{e}(\mathsf{CloseToDoor}), \Downarrow\mathsf{e}(\mathsf{InCorridor})\})[t_1, t_2] \\ \wedge\ & AG(\{\mathsf{Hi}(\mathsf{InCorridor})\})[t_0, t_1] \\ \wedge\ & AG(\{\mathsf{Lo}(\mathsf{TimeOut})\})[t_0, t] \end{aligned} \tag{15}$$

Assuming reasonable settings of the Hi and Lo thresholds h, l, the following substitutions of the time variables to time-points yield the mapping into the activation histories in Fig. 3: $t = 28$ (right outside the figure), $t_0 = 3, t_1 = 12, t_2 = 16, t_3 = 20, t_4 = 24$. As a result, we extract InRoom$[24]$.

This substitution is not unique. For example, postponing t_0 until 5 or having t_1 earlier at 9 would also work. This point leads to the process of *chronicle recognition:* given a working BBS, permanently producing activation values, how are the given defining chronicles of facts checked against that activation value data stream to determine whether some fact starts to hold?

The obvious basis for doing this is to keep track of the qualitative activations as they emerge. That means, for every behavior, there is a process logging permanently the qualitative activations. For those of type Hi and Lo, the sufficiently long time periods of the respective behavior activation above and below the h, l thresholds, resp., have to be recorded and, if adjacent to the current time point,

appropriately extended. This would lead automatically to identifying qualitative activations of types Hi and Lo with their earliest start point, such as $t_0 = 3$ for Hi(InCorridor) in the example above. Qualitative activations of types ⇑e and ⇓e are logged iff their definitions (eqs. 13 and 14, resp.) are fulfilled in the recent history of activation values.

Qualitative activation logs are then permanently analyzed whether any of the existing defining chronicles are fulfilled, which may run in parallel to the ongoing process of logging the qualitative activations. An online version of this analysis inspired by [Gha96] would attempt to match the flow of qualitative activations with all defining chronicles c by means of *matching fronts* that jump along c's internal interval boundary points t_i and try to bind the next time point t_{i+1} as current matching front such that the recent qualitative activations fit all sub-intervals of c that end in t_{i+1}. Note that more than one matching front may be active in every defining chronicle at any time. A matching front in c vanishes if it reaches the end point t (the defining chronicle is true), or else while stuck at t_i is caught up by another matching front at t_i, or else an activation gestalt over an interval ending at t_{i+1} is no longer valid in the current qualitative activation history. The complexity of this process is O(|Behaviors|*|Chronicles|*max|AGsInChronicle|), see [SCHC01] for details.

Practically, the necessary computation may be focused by specifying for each defining chronicle a *trigger condition,* i.e., one of the qualitative activations in the definition that is used to start a monitoring process of the validity of all activation gestalts. For example, in the InRoom definition above, ⇑e(GoThruDoor), as occurring in the $[t_3, t_4]$ interval, might be used. Note that the trigger condition need not be part of the earliest activation gestalts in the definition. On appearance of some trigger condition in the qualitative activation log, we try to match the activation gestalts prior to the trigger with qualitative activations in the log file, and, if successful, verify the gestalts after the trigger condition in the qualitative activations as they are being logged. [SCHC01] gives an example where the derivation of the InRoom fact fails and a trigger condition would save unnecessary matching effort.

Some more general remarks are in place here. Our intention is to provide fact extraction from activation values as a *main* source of information, not the exclusive one. In general, the obvious other elements may be used for defining them: sensor readings at some time points (be they physical sensors or sensor filters), and the validity of symbolic facts at a time point or over some time interval. That type of information can be added to the logical format of a chronicle definition as in (15). For example, if the exact time point of entering a room with the robot's front is desired as the starting point of the InRoom fact, then this might be determined by the time within the interval $[t_4, t]$ (i.e., within the decrease of the GoThruDoor activation) where some sensor senses open space to the left and right again. As another example, assume that the fact At($Door_A$) for the door to some room A may be in the fact base. Then At($Door_A$)$[t_4]$ could be added to the defining chronicle (15) above to derive not only InRoom$[t]$, but more specifically InRoom(A)$[t]$. Such a position information can be easily derived from

a normal localization process like [Fox01]. This kind of knowledge base update would be purely sensor related and orthogonal to chronicle recognition. This mechanism can also be used for "instanciated actions" like **GO-IN-RM**(A). Its preconditions mentioned at the end of Sec. 5 would be generated by the planner's knowledge base rule $\mathsf{At}(x) \Rightarrow \mathsf{CloseTo}(x)$.

The fact extraction technique does not presume or guarantee anything about the consistency of the facts that get derived over time. Achieving and maintaining consistency, and determining the ramifications of newly emerged facts remain issues that go beyond fact extraction. Pragmatically, we would not recommend to blindly add a fact as true to the fact base as soon as its defining chronicle has been observed. A consequent of a recognized defining chronicle should be interpreted as evidence for the fact or as a fact *hypothesis,* which should be added to the robot's knowledge base only by a more comprehensive knowledge base update process, which may even reject the hypothesis in case of conflicting information. A possible solution would be to add some integrity constraints to the defining chronicles, like adding InRoom(A) means deleting all InRoom(y) with $y \neq A$. However, this is not within the scope of this paper.

7 Discussion

A physical agent's perception categories must to some degree be in harmony with its actuator capabilities—at least in purposively designed technical artifacts such as working autonomous robots.[1] Our approach of extracting symbolic facts from behavior activation merely exploits this harmony for intertwining control on a symbolic and a reactive level of a hybrid robot control architecture.

The technical basis for the exploitation are time series of behavior activation values. We have taken them from a special type of behavior-based robot control systems (BBSs), namely, those consisting of behaviors expressed by nonlinear dynamical functions of a particular form, as described in Sec. 2. The point of having activation values in BBSs is not new; it is also the case, e.g., for the behavior-based fuzzy control part underlying Saphira [KMSR97], where the activation values are used for context-dependent *blending* of behavior outputs, which is similar to their use in our BBS framework. Activation values also provide the degree of applicability of the corresponding motor schemas in [Ark98, p. 141].

The activation values of a dynamical system-type BBS are well-suited for fact extraction in that their formal background in dynamical systems theory provides both the motivation and the mathematical inventory to make them change smoothly over time—compare, e.g., the curves in Figure 3 with the ragged ones in [SRK99, Fig. 5.10]. This typical smoothness is handy for defining *qualitative activations,* which aggregate particular patterns in terms of edges and levels of the curves of individual behaviors, which are recorded as they emerge over time. These then serve as a stable basis for chronicle recognition over qualitative activations of several behaviors. Note, however, that this smoothness is a practical

[1] We do not speculate about biological agents in this paper, although we would conjecture that natural selection and parsimony strongly favor this principle.

rather than a theoretical issue, and other BBS approaches may serve as bases for fact extraction from activation values.

Using the system dynamics itself as information source seems to be a promising idea even in the area of cognitive psychology. Making the program sensitive to patterns in its own processing, the so called *self-watching*, is an interesting commonality between our ideas and the approach described by Marshall in [Mar99]. However, the targeted application area differs completely from robot control: Marshall computationally models the fundamental mechanisms underlying human cognition on analogy-making in ASCII-strings. In this microdomain, self-watching helps to understand how an answer was found and to detect senseless looping in the system. The latter could be a interesting extension for our approach.

We want to emphasize that the activation values serve three purposes in our case: first, their normal one to provide a reliable BBS, second, to deliver the basis for extracting persistent facts, and third, as an interface for biasing behaviors towards the current symbolic action's intention. With the second use, we save the domain modeler a significant part of the burden of designing a complicated sensor interpretation scheme only for deriving facts. The behavior activation curves, as a by-product coming for free of the behavior-based robot control, focus on the environment dynamics, be it induced by the robot itself or externally. By construction, these curves aggregate the available sensor data in a way that is particularly relevant for robot action. We have argued that this information can be used as a main source of information about the environment; other information, such as coming from raw sensor data, from dedicated sensor interpretation processes, or from available symbolic knowledge, could and should be used in addition.

As activation values are present in a BBS anyway, it is possible to "plug-in" the deliberative component to an already existing behavior system like the DD control system in [BGG+99]. Yet, if a new robot control system is about to be written for a new application area, things could be done better, within the degrees of freedom for variations in behavior and domain model design. The ideal case is that the behavior inventory and the fact set is in harmony in the sense that such facts get used in the domain model whose momentary validity engraves itself in the activation value history, and such behaviors get used that produce activation values producing evidence for facts. For example, a single WallFollow behavior working for walls on the right and on the left, may be satisfactory from the viewpoint of behavior design for a given robot application. For the deliberative part, it may be more opportune to split it into FollowLeftWall and FollowRightWall, which would be equally feasible for the behavior control, but allows more targeted facts to be deduced and behaviors to be biased directly. The fact extraction and behavior biasing scheme leaves the possibility to unplug the deliberative part from the robot control, which we think is *essential* for robustness of the whole robot system.

Our technique is complementary to anchoring symbols to sensor data as described in [CS01]. It differs from that line of work in two main respects. First,

we use sensor data as aggregated in activation value histories only, not raw sensor data. Second, we aim at extracting ground facts rather than establishing a correspondence between percepts and references to physical objects. The limit of our approach is that it is inherently robot-centered in the sense that we can only arrive at information that has to do directly with the robot action. The advantage is that, due to its specificity, it is conceptually and algorithmically simpler than symbol anchoring in general.

Behavior(-sub)systems have been modeled as self-organizing dynamical systems by other researchers, too. Steinhage presents an approach where the control architecture is very strongly related to dynamical systems theories. Even position estimation, sensor fusion, trajectory planning, event counting, or behavior selection is entirely described in terms of differential equations, see e.g. [SS98] for details. From our point of view, using a planner instead of coding all goal directed knowledge directly into the BBS makes it much easier to define and change "high-level" mission goals and to communicate plans between agents. Most robot control architecture used a kind of situation depending *context* – called context-dependent blending [SRK99] or sensor context [SS98] – which helps regulating the behavior activation. DD&P supports this context – via activation value biasing – in a very flexible way without "touching" the BBS description. Validation of the overall architecture is even easier with a clear view on the operator that is currently being executed.

DD&P permits users not familiar with BBS and robot control to define new tasks for our robots *only* with formal logic. Following the principle of "Plan-as-Advice", a well-defined BBS and influence-matrix would even not allow the user to drive the robot into a wall or to break down with empty batteries. This kind of effect can be even used for *opportunistic* task execution. In a mail-delivery example, a person X that the robot meets by chance on the corridor, can be handed over a letter directly, instead of following the strict plan sequence which would lead – among others – to the office of X. These effects can result from causal links in the current plan – similar to STRIPS's triangle tables [FHN72] – where an operator is skipped if its relevant postconditions are already valid. *Opportunistic* mail-delivery to a person Y that is not included in the current plan can only be achieved by pre-defined conditions.

The fact extraction technique can also be used to monitor the execution of actions. With our example in Sec. 6 the execution of the symbolic action EnterNextRoom can be monitored, simply by means of post-conditions of actions, like InRoom in this case.

Our approach is not in principle limited to a particular combination of deliberation component and BBS, as long as the BBS is expressed as a dynamical system and involves a looping computation of activation values for the behaviors. The method for fact extraction from activation value histories of BBSs presented here is potentially applicable in BBS frameworks other than DD, so long as they are smooth enough to allow some instance of chronicle recognition.

8 Conclusion

We have presented a new approach for extracting information about symbolic facts from activation curves in behavior-based robot control systems. To this end, we use already available, aggregated, filtered, and fused sensor-values instead of re-calculating this data a second time in the symbol grounding component.

Updating the symbolic environment situation is a crucial issue in hybrid robot control architectures in order to bring to bear the reasoning capabilities of the deliberative control part on the physical robot action as exerted by the reactive part. Unlike standard approaches to sensing the environment in robotics, we are using the information hidden in the temporal development of the data, rather than their momentary values. Therefore, our method promises to yield environment information that is complementary to normal sensor interpretation techniques, which can and should be used in addition.

Biasing behaviors instead of exerting hard control fits nicely into a general architectural design rule: Focusing action execution always on the *current* state of the world. Additionally it offers the chance of opportunistical reactions to unforeseen events.

We have presented these techniques in principle as well as in terms of selected demo examples in a robot simulator, which has allowed to judge the approach feasible and to design the respective algorithms.

Work is ongoing towards a physically concurrent implementation of DD&Pon physical robots, as described in [HS01]. We have recently implemented an alternative approach for fact extraction based on supervised learning [JHS02]. At the present time, we are extending and improving the tools that are available in the context of DD&P. We are confident that a complex software system, like a robot control architecture, can only get over the prototype stage if a proper programming environment is available that supports its general applicability.

Acknowledgments

Thanks to Mihaela Cistelecan, Herbert Jaeger, Hans-Ulrich Kobialka, and the Fraunhofer AiS.BE-Team for their support on this work.

References

[Age] The AgenTec project. `http://www.agentec.de`.

[Ark98] R. Arkin. *Behavior-Based Robotics*. MIT Press, 1998.

[BFG+97] P. Bonasso, J. Firby, E. Gat, D. Kortenkamp, D. Miller, and M. Slack. Experiences with an architecture for intelligent, reactive agents. *Journal of Experimental and Theoretical Artificial Intelligence: JETAI*, 9:237–256, 1997.

[BGG+99] A. Bredenfeld, W. Göhring, H. Günter, H. Jaeger, H.-U. Kobialka, P.-G. Plöger, P. Schöll, A. Siegberg, A. Streit, C. Verbeek, and J. Wilberg. Behavior engineering with *dual dynamics* models and design tools. In *Proc. 3rd Int. Workshop on RoboCup at IJCAI'99*, pages 57–62, 1999.

[BK01] A. Bredenfeld and H.-U. Kobialka. Team cooperation using dual dynamics. In *Balancing Reactivity and Social Deliberation in Multi-Agent Systems*, Volume 2103 in LNAI, pages 111–124. Springer, 2001.

[Bre00] A. Bredenfeld. Integration and evolution of model-based tool prototypes. In *11th IEEE International Workshop on Rapid System Prototyping*, pages 142–147, Paris, France, June 2000.

[CS01] S. Coradeschi and A. Saffiotti. Perceptual anchoring of symbols for action. In *Proc. of the 17th IJCAI Conf.*, pages 407–412, Seattle, WA, August 2001.

[FBT99] D. Fox, W. Burgard, and S. Thrun. Markov localization for mobile robots in dynamic environments. *J. Artif. Intell. Research (JAIR)*, 11, 1999.

[FHN72] R.E. Fikes, P.E. Hart, and N.J. Nilsson. Learning and executing generalized robot plans. *Artif. Intell.*, 3:251–288, 1972.

[FL01] M. Fox and D. Long. PDDL2.1: An extension to PDDL for expressing temporal planning constraints. `http://www.dur.ac.uk/d.p.long/pddl2.ps.gz`, 2001.

[Fox01] D. Fox. KLD-Sampling: Adaptive particle filters. In *Advances in Neural Information Processing Systems 14*. MIT Press, 2001.

[Gha96] M. Ghallab. On chronicles: Representation, on-line recognition and learning. In Aiello, Doyle, and Shapiro, editors, *Proc. Principles of Knowledge Representation and Reasoning (KR'96)*, pages 597–606. Morgan-Kauffman, November 1996.

[Har90] S. Harnad. The symbol grounding problem. *Physica D*, 42:335–346, 1990.

[HJZM98] J. Hertzberg, H. Jaeger, U. Zimmer, and Ph. Morignot. A framework for plan execution in behavior-based robots. In *Proc. of the 1998 IEEE Int. Symp. on Intell. Control (ISIC'98)*, pages 8–13, Gaithersburg, MD, September 1998.

[HS01] J. Hertzberg and F. Schönherr. Concurrency in the DD&P robot control architecture. In M.F. Sebaaly, editor, *Proc. of The Int. NAISO Congr. on Information Science Innovations (ISI'2001)*, pages 1079–1085. ICSC Academic Press, march, 17–21 2001.

[JC97] H. Jaeger and Th. Christaller. Dual dynamics: Designing behavior systems for autonomous robots. In S. Fujimura and M. Sugisaka, editors, *Proc. Int. Symposium on Artificial Life and Robotics (AROB'97)*, pages 76–79, 1997.

[JHS02] H. Jaeger, J. Hertzberg, and F. Schönherr. Learning to ground fact symbols in behavior-based robots. In F. van Harmelen, editor, *Proceedings of the 15th European Conference on Artificial Intelligence (ECAI 2002)*, Amsterdam, In Press, 2002. IOS Press.

[KJ02] H.-U. Kobialka and H. Jaeger. Experiences using the dynamical system paradigm for programming robocup robots. unpublished, 2002.

[KMSR97] K. Konolige, K. Myers, A. Saffiotti, and E. Ruspini. The Saphira architecture: A design for autonomy. *Journal of Experimental and Theoretical Artificial Intelligence: JETAI*, 9:215–235, 1997.

[KNHD97] J. Koehler, B. Nebel, J. Hoffmann, and Y. Dimopoulos. Extending planning graphs to an ADL subset. In S. Steel and R. Alami, editors, *Recent Advances in AI Planning. ECP'97*, pages 273–285. Springer (LNAI 1348), 1997. `http://www.informatik.uni-freiburg.de/~koehler/ipp.html`.

[Mar99] J. Marshall. *Metacat: A Self-Watching Cognitive Architecture for Analogy-Making and High-Level Perception.* PhD thesis, Indiana University, Bloomington, IN, USA, November 1999.

[Mat99] M.J. Matarić. Behavior-based robotics. In Robert A. Wilson and Frank C. Keil, editors, *MIT Encyclopedia of Cognitive Sciences*, pages 74–77. The MIT Press, April 1999.

[MNPW98] N. Muscettola, P. Nayak, B. Pell, and B.C. Williams. Remote Agent: to boldly go where no AI system has gone before. *J. Artificial Intelligence*, 103:5–47, 1998.

[NCLMA99] D.S. Nau, Y. Cao, A. Lotem, and H. Munoz-Avila. SHOP: Simple hierarchical ordered planner. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI'99)*, pages 968–975. Morgan Kaufmann Publishers, 1999.

[NDK97] B. Nebel, Y. Dimopoulos, and J. Koehler. Ignoring irrelevant facts and operators in plan generation. In *ECP'97*, pages 338–350, 1997.

[Pol92] M.E. Pollack. The uses of plans. *Artif. Intl.*, 57(1):43–68, 1992.

[PRK90] D.W. Payton, J.K. Rosenblatt, and D.M. Keirsey. Plan guided reaction. *IEEE Transactions on System, Man, and Cybernetics*, 20(6):1370–1382, 1990.

[Rem00] Remote Agent Project. Remote agent experiment validation. `http://rax.arc.nasa.gov/DS1-Tech-report.pdf`, February 2000.

[Ros97] J.K. Rosenblatt. DAMN: A distributed architecture for mobile navigation. *Journal of Experimental and Theoretical Artificial Intelligence*, 9(2/3):339–360, 1997.

[SCHC01] F. Schönherr, M. Cistelecan, J. Hertzberg, and Th. Christaller. Extracting situation facts from activation value histories in behavior-based robots. In F. Baader, G. Brewka, and T. Eiter, editors, *KI 2001: Advances in Artificial Intelligence (Joint German/Austrian Conference on AI, Proceedings*, volume 2174 of *LNAI*, pages 305–319. Springer, 2001.

[SKR95] A. Saffiotti, K. Konolige, and E.H. Ruspini. A multivalued-logic approach to integrating planning and control. *Artificial Intelligence*, 1995.

[SRK99] A. Saffiotti, E.H. Ruspin, and K. Konolige. Using fuzzy logic for mobile robot control. In H-J. Zimmermann, editor, *Practical Applications of Fuzzy Technologies*, chapter 5, pages 185–205. Kluwer Academic, 1999. Handbook of Fuzzy Sets, vol.6.

[SS98] A. Steinhage and G. Schöner. Dynamical systems for the behavioral organization of autonomous robot navigation. In P.S. Schenker and G.T. McKee, editors, *Sensor Fusion and Decentralized Control in Robotic Systems: Proceedings of SPIE*, volume 3523, pages 169–180, 1998.

[Suc87] L.A. Suchman. *Plans and Situated Action: The Problem of Human-Machine Communication*. Cambridge University Press, 1987.

Decision-Theoretic Control of Planetary Rovers

Shlomo Zilberstein[1], Richard Washington[2],
Daniel S. Bernstein[1], and Abdel-Illah Mouaddib[3]

[1] Univ. of Massachusetts, Dept. of Computer Science, Amherst, MA 01003, USA
[2] RIACS, NASA Ames Research Center, MS 269-3, Moffett Field, CA 94035, USA
[3] Laboratoire GREYC, Université de Caen, F14032 Caen Cedex, France

Abstract. Planetary rovers are small unmanned vehicles equipped with cameras and a variety of sensors used for scientific experiments. They must operate under tight constraints over such resources as operation time, power, storage capacity, and communication bandwidth. Moreover, the limited computational resources of the rover limit the complexity of on-line planning and scheduling. We describe two decision-theoretic approaches to maximize the productivity of planetary rovers: one based on adaptive planning and the other on hierarchical reinforcement learning. Both approaches map the problem into a Markov decision problem and attempt to solve a large part of the problem off-line, exploiting the structure of the plan and independence between plan components. We examine the advantages and limitations of these techniques and their scalability.

1 Towards Autonomous Planetary Rovers

The power of a mobile platform to perform science and explore the surface of distant planetary surfaces has long attracted the attention of the space exploration community. Unmanned rovers have been deployed on the Moon and on Mars, and they have been proposed for exploring other planets, moons, and small bodies such as asteroids and comets. The challenges and goals of planetary exploration pose unique constraints on the control of rovers, constraints that differentiate this domain from others that have traditionally been considered in mobile robotics. In addition, operation of a rover on a planetary surface differs significantly from operation of other distant spacecraft.

In this paper, we describe the problem of rover control and illustrate its unique aspects. We show how these characteristics have led us to consider utility as a fundamental concept underlying planetary exploration; this in turn directed our attention and effort to decision-theoretic approaches for planetary rover control. We will survey these approaches, particularly concentrating on two methods: one based on adaptive planning and the other on hierarchical reinforcement learning.

A planetary rover is first and foremost a science tool, carrying a suite of instruments to characterize a distant environment and to transmit information to Earth. These instruments may include cameras, spectrometers, manipulators,

M. Beetz et al. (Eds.): Plan-Based Control of Robotic Agents, LNAI 2466, pp. 270–289, 2002.

and sampling devices. Under some level of control from Earth-bound scientists and engineers, the rover deploys the instruments to gain information about the planetary surface. For example, in the Mars Smart Lander mission, currently planned for 2009, a rover will traverse a few kilometers between scientifically interesting sites. At each site, the rover will visit a number of targets (typically rocks) and deploy instruments on each one. In the current mission scenario, the targets and rover actions will be completely specified by scientists and rover engineers. The work presented in this paper would enable the rover to perform many of these steps autonomously.

The level of success of a rover mission is measured by the "science return," or amount of useful scientific data returned to the scientists on Earth. Although it is difficult to measure concretely, some attempts have been made to characterize it precisely for particular scenarios [30]. Criteria such as rover safety, navigation accuracy and speed, data compression ratios, and resource management contribute to science return. An important characteristic of using science return as a mission success criterion is that it is a quantity to be maximized, not a discrete goal to be achieved. This differs markedly from traditional applications of planning technology to mobile robotics. From the early days of planning, applications to robotics have typically concentrated on achieving discrete goals [18,15,28]. More recently, decision-theoretic planning has extended beyond all-or-none goals to handle overall reward [21,29], offering a more suitable framework for planetary rover control.

Autonomous control of rovers on distant planets is necessary because the round-trip time for communication makes tele-operation infeasible. Many earth-based rovers, as well as lunar rovers to a certain extent, can be controlled via tele-operation, using advanced user interfaces to compensate for latency in communication links [11,1]. For Martian or other distant planetary exploration, the latency increases beyond the limits of tele-operation. In addition, because of constraints on communication resources and cost, currently envisioned missions will limit communications to once or twice daily. Between these communication opportunities, the rover must operate autonomously.

An important and distinctive feature of planetary robotics, and a challenge for autonomous operations, is uncertainty. With planetary rovers, there is uncertainty about many aspects of sequence execution: exactly how long operations will take, how much power will be consumed, and how much data storage will be needed. Resources such as power and data storage are critical limits to rover operations; resource limits must be respected, but unused resources generally translate to wasted mission time and thus decreased productivity. Furthermore, there is uncertainty about environmental factors that influence such things as rate of battery charging or which scientific tasks are possible. In order to allow for both sources of uncertainty, a traditional spacecraft command plan is conservative: only limited operations are allowed within a single uplink, time and resource usage are based on worst-case estimates, and the plan contains fail-safe checks to avoid resource overruns. If an operation takes less time than expected, the rover waits until the time prescribed for the next operation. If an

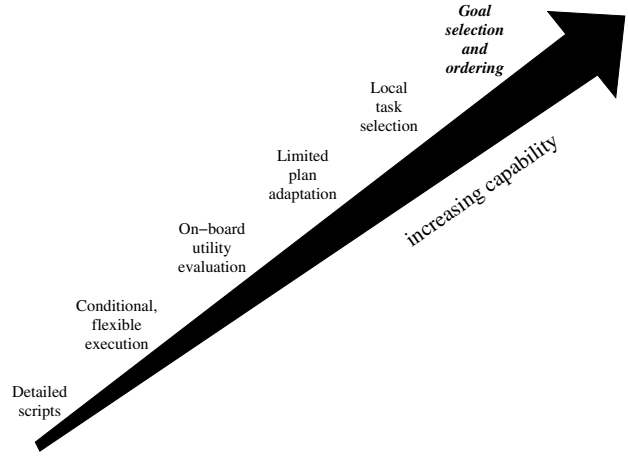


Fig. 1. Increasing levels of capability for planetary rovers

operation takes longer than expected, it may be terminated before completion; in some cases, all non-essential operations may be halted until a new command plan is received. These situations result in unnecessary delays and lost science opportunities.

An example is the Mars Smart Lander mission, where the rover will visit at most one target in a single uplink, and in fact the rover will only approach a target and place an instrument before waiting for the next command plan [22]. Although conservative, this is still an advance over previous rovers (Sojourner [23] or the 2003 Mars Exploration Rovers), which required multiple days to accomplish as much. The techniques described in this paper provide the rover with the ability to select and balance tasks across multiple targets, allowing more ambitious exploration.

The highly uncertain operational environment distinguishes rover control from other spacecraft control. A deep space probe works in a harsh but stable environment, and its actions have relatively predictable effects, barring anomalies. Planning procedures designed for spacecraft [25,17] do not explicitly handle all the types of uncertainty; applications of these technologies to the problem of rover control [14] rely on the presence of planners on board to replan when execution diverges from a single nominal plan.

The computational power of planetary rovers is also severely limited by the use of radiation-hardened, low-power processors and electronics. Increases in processor performance are more than made up for by the desire for increased on-board processing of images and science data, as well as improved navigation. In addition, the processor is a draw on the overall power budget. Thus control approaches that minimize on-board computation are preferable.

Constrained by action and environmental uncertainty, and limited computational resources, our objective is to increase the science productivity possible

Fig. 2. The K9 Rover

within a single uplink. To this end, we are pursuing a program of increasing capabilities, illustrated in Figure 1. Starting from the capabilities of the Sojourner rover, which used detailed, time-stamped scripts of low-level commands, we are moving toward autonomous goal selection and ordering. The latter is the main focus of this paper. Before presenting that work, we first review the steps along this spectrum of capabilities.

In all past and currently planned missions, the command plans for the rover are completely specified on the ground. In this case, additional flexibility in terms of time and state conditions, as well as contingent branches, may allow a wider range of behaviors than fixed-time sequences [6]. Additional capability can be realized by calculating utilities of plan branches with respect to the situation at execution time [7]. If we allow limited innovation on board, the rover can adapt its plan to the situation by skipping steps or merging in plan fragments from a plan library constructed and verified on the ground [8].

The capabilities described to this point make use of plans that have been pre-specified to full detail. To specify plans at a higher level of abstraction, such as desired science targets, the decomposition of the high-level tasks into detailed actions must be performed on board in a way that is sensitive to the execution context. Decision-theoretic planning and control methods can perform this dynamic choice to maximize science return.

If science targets are considered individually, the problem is a local problem of *task selection*. The control problem in this case is to decide which experiments

to perform and when to move to another target [3]. The latter depends on the expected information to be gained from other targets and the difficulty of reaching them. The former depends on the available resources as well as characteristics of the target.

Alternatively, we may reason about a group of targets together; in planetary exploration this is often referred to as a *site*. By considering the activities within a site together, the overall science return can be improved compared to target-specific control policies. It is at this level of capability that we concentrate for the remainder of the paper.

The planning and execution techniques described in this paper allow the rover to re-prioritize and reorder scientific activities based on progress made, scientific observations, and the success or failure of past activities. The solution relies on off-line analysis of the problem and on pre-compilation of control policies. In addition, we have used the independence between various mission tasks and goals to reduce the complexity of the control problem. The rest of the paper is organized as follows. Section 2 sketches the multiple layers of control and the way the decision-theoretic planning component interacts with the lower-levels. The section also provides a general introduction to decision-theoretic control and the two approaches we have developed. These approaches, adaptive planning and hierarchical reinforcement learning are detailed in sections 3 and 4. We conclude with a discussion of the merits of these two approaches and future work.

2 Layers of Control

The focus of this paper is on high-level, decision-theoretic control. However, the decision-theoretic component does not interact directly with the rover's actuators. It rests on a number of existing layers of control, which bridge the gap between decision-theoretic plans and the low-level control of the robotic mechanisms. In this section we describe the entire control architecture and the experimental platform for which it has been developed.

We are targeting our work for the NASA Ames "K9" rover prototype, pictured in Figure 2. The existing rover software architecture in place on the K9 rover consists of four distinct layers, as shown in Figure 3. Low-level device drivers communicate with hardware. Mid-level component controllers receive simple commands (such as direct movement, imaging, and instrument commands) and communicate with the device drivers to effectuate the commands. Abstract commands implement compound or complex actions (such as movement with obstacle avoidance, visual servoing to a target, and arm placement). A plan executive interprets command plans and calls both simple and abstract commands as specified in the plan. For more details on this architecture, see [9].

A high-level, decision-theoretic controller interacts with this architecture by proposing high-level actions, potentially more abstract than commands within the architecture. The high-level actions are then decomposed into small command plans; these command plans are provided to the rover plan executive, which in turn manages the execution and monitoring of the low-level commands

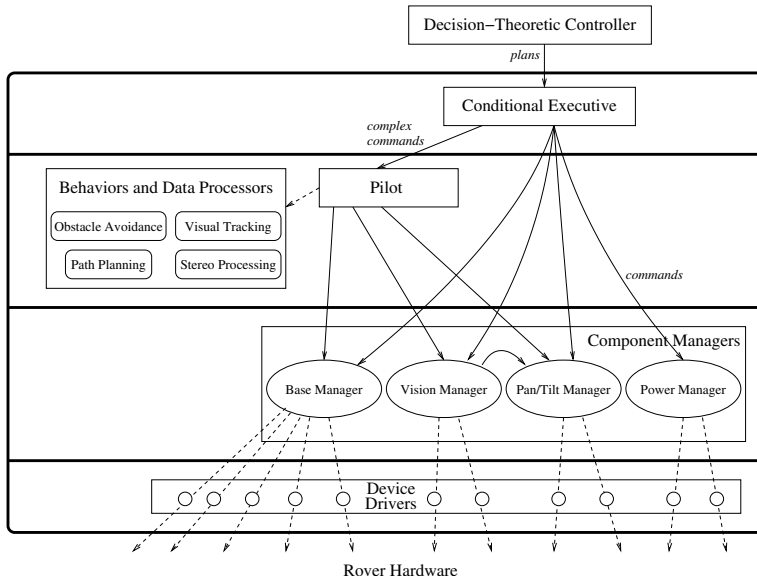


Fig. 3. Layers of rover software in the existing K9 rover software architecture

within the command plans. Information about success of the high-level action and the resulting state of the system is returned to the decision-theoretic controller at the end of execution of each command plan.

The rover control problem, at the level that we are addressing, consists of a set of science-related goals. These science goals identify a set of targets, each of which has particular scientific interest. The rover has a set of instruments available, and thus a set of possible experiments, to gather relevant information about the targets. Given the set of targets and desired information, the rover's task is to choose activities that provide the maximum information possible about the targets within resource and time constraints and to return that information to the scientists.

3 Decision-Theoretic Control

The high-level control of the rover presents a sequential decision problem under uncertainty. At each point, the rover must select the next action based on the current state and the remaining plan. The state in this case includes both features characterizing the environment and features characterizing the rover itself, such as an indication of the remaining resources. Such problems can be modeled as a *Markov decision process* (MDP), assuming that each action transforms the current state into one of several possible outcome states with some fixed transition probability. This assumption is also referred to as the Markov assumption.

More formally, an MDP is defined by a finite set of *states*, S; a finite set of possible *actions*, A; and a *transition probability* function $Pr(s'|s,a)$ that indicates the probability that taking action $a \in A$ in state $s \in S$ results in a transition to state $s' \in S$. Each transition has an associated *reward*, $R(s,a)$, which can capture the cost of the action, the value of the outcome, or some combination of both. The objective is to maximize the reward over a finite- or infinite-horizon. In the later case, future reward is typically discounted by a factor of γ^i, where i is the number of steps. Partially-observable MDPs (or POMDPs) generalize the MDP model by allowing the agent to have only partial information about the state. At the end of each action, the agent can make an observation $o \in \Omega$. A belief function over states can be maintained using Bayesian updating given the *observation probability* function, $Pr(o|s,a)$. (In general, both transition probabilities and observation probabilities may also depend on the outcome state.) We limit our discussion in this paper to MDPs, which provide adequate model for high-level rover control.

A solution to an MDP can be represented as a mapping from states to actions, $\pi : S \rightarrow A$, called a *policy*. Several dynamic programming algorithms (such as value iteration and policy iteration) have been developed for finding optimal control policies for MDPs [27,31]. It has also been shown that MDPs can be solved using heuristic search by such algorithms as LAO* [19]. The advantage heuristic search has over dynamic programming is that, given an initial state, it can find an optimal solution without evaluating the entire state space. Dynamic programming, in contrast, evaluates the entire state space, finding a policy for every possible starting state. For problems with large state spaces, heuristic search offers substantial computational savings.

One important characteristic of the rover control problem is the explicit modeling of the amount of resources action use. Extensions of MDPs to handle duration of actions have been previously studied. For example, in Semi-Markov Decision processes (SMDPs) actions can have stochastic durations and state transitions are stochastic [27]. In Stochastic Time Dependent Networks (STDNs) actions can have stochastic durations, but state transitions are deterministic [34]. In Time-Dependent MDPs (TMDPs) actions can have stochastic, time-dependent durations and state transitions are stochastic [5]. The model we use in this paper have both stochastic state transitions and actions that consume varying levels of resources (not just time). In this sense, the model is a proper extension of the previous ones. While modeling the consumption of resources by actions is not difficult, it increases the state space dramatically. The number of states grow linearly with the number of resource units (which could be large) and exponentially with the number of resources. However, resources have additional characteristics that simplify their treatment: all the units of a resource are typically exchangeable, the number of units goes down as they are consumed (unless the resource is renewable), and the amount of resources used by an action typically depends only on the action itself. Therefore, treating resources as just any other component of the state is wasteful.

In the following two sections we examine two decision-theoretic techniques that take advantage of the unique characteristics of the rover control problem in order to simplify it. both approaches are based on modeling the rover set of activities as a loosley-coupled MDP. The scientific experiments the rover performs in each location are largely independent, but they share the same resources. The first approach develops a local policy for each activity that takes into account the remaining plan by computing a cost function over resources. By estimating quickly this cost function at run-time, we can avoid solving the entire MDP while producing near-optimal control policies. The second approach is based on a hierarchical reinforcement learning algorithm designed to take advantage of the natural decomposability offered by loosely-coupled MDPs. It maintains two different value functions: a low-level state-action value function defined over all state-action pairs and a high-level state value function defined only over "bottleneck" states that bridge the components of the MDP.

The two approaches share the ability to accelerate policy construction by exploiting the structure of the MDP, but they offer different advantages and disadvantages. The first approach exploits a model of the domain and allows for off-line policy construction and compact policy representation, both important issues in rover control. The second approach is model free and is particularly suitable for operation in poorly modeled environments or for adaptation of an existing policy to new environments. The next two sections describe the two approaches and examine their characteristics.

4 Adaptive Planning Approach

The adaptive planning approach is based on off-line analysis of each possible rover activity and construction of policies for each possible activity using dynamic programming. The key question is how to adapt pre-compiled policies at run-time to reflect the dynamic execution state of the plan. The dynamic information includes the remaining workload and the remaining resources, both of which can be captured by the notion of *opportunity cost*.

Each *plan* assigned to a rover is composed of a sequence of target *activities* represented as progressive processing task structures [24,35]. An initial resource allocation is also specified. Resources are represented as vectors of discrete units. We assume here that the plan is totally ordered and that resources are not renewable. A generalization of the technique to acyclic graphs has been examined in [10].

4.1 The Rover Model

The rover can perform a certain set of predefined activities, each of which has an associated fixed task structure. The task structure is represented as a *progressive processing unit* (PRU), which is composed of a sequence of *steps* or *processing levels*, $(l_1, l_2, \ldots)$. Each step, l_i, is composed of a set of *alternative modules*, $\{m_i^1, m_i^2, \ldots\}$. Each module of a given step can perform the same logical function,

but it has different computational characteristics defined by its *descriptor*. The module descriptor, $P_i^j((q', \Delta r)|q)$, of module m_i^j is the probability distribution of output quality and resource consumption for a given input quality. Module descriptors are similar to *conditional performance profiles* of anytime algorithms.

When the rover completes an activity, it receives a reward that depends on the quality of the output and the specific activity. Each PRU has an associated *reward function*, $U(q)$, that measures the immediate reward for performing the activity with overall quality q. Rewards are cumulative over different activities.

Given a plan, a library of task structures that specify a PRU for each activity in the plan, the module descriptors of all the components of these PRUs, and corresponding reward functions for each activity, we want to select the best set of alternative modules to maximize the overall utility or scientific return of the rover.

4.2 Optimal Control of a Single Activity

We begin with the problem of meta-level control of a single progressive processing unit corresponding to a single activity. This problem can be formulated as a Markov decision process (MDP) with states representing the current state of the activity. The state includes the current level of the PRU, the quality produced so far, and the remaining resources. The rewards are defined by the utility of the solution. The possible actions are to *execute* one of the modules of the next processing level. The transition model is defined by the descriptor of the module selected for execution.

State Transition Model. The execution of a single progressive processing unit can be seen as an MDP with a finite set of states $\mathcal{S} = \{[l_i, q, r]\}$, where i indicates the last executed level, q is the quality produced by the last executed module, and r is the remaining resources. When the system is in state $[l_i, q, r]$, one module of the i-th level has been executed. (The first level is $i = 1$; $i = 0$ is used to indicate the fact that no level has been executed.)

The initial state of the MDP is $[l_0, 0, r]$, where r is the available resources for plan execution. (Additional resources may be reserved for rover operation once execution of the plan is complete.) The initial state indicates that the system is ready to start executing a module of the first level of the PRU. The terminal states are all the states of the form $[l_L, q, r]$, where L is the last level of the PRU. In particular, the state $[l_L, 0, r]$ represents termination with no useful result and remaining resources r. Termination with $r = 0$ is also possible; if that happens during the execution of an intermediate level of the PRU, a last transition is made to the end of the PRU by skipping all the levels that cannot be performed.

In every nonterminal state, $[l_i, q, r]$, the possible actions, designated by $\mathbf{E}_{i+1}^j$, execute the j-th module of the next level. The outcome of action $\mathbf{E}_{i+1}^j$ is probabilistic. Resource consumption and quality uncertainties define the new state as follows.

$$Pr([l_{i+1}, q', r - \Delta r] \mid [l_i, q, r], \mathbf{E}_{i+1}^j) = P_{i+1}^j((q', \Delta r)|q) \quad (1)$$

Rewards and the Value Function. Rewards are determined by the given reward function applied to the final outcome. Note that no rewards are associated with intermediate results, although this could be easily incorporated into the model. The value function (expected reward-to-go) over all states is defined as follows. The value of a terminal state is based on the utility of the results.

$$V([l_L, q, r]) = U(q) \quad (2)$$

The value of a nonterminal state of the MDP is defined as follows.

$$V([l_i, q, r]) = \max_j \sum_{q', \Delta r} P_{i+1}^j((q', \Delta r)|q)\ V([l_{i+1}, q', r - \Delta r]) \quad (3)$$

This concludes the definition of a finite-horizon MDP, or equivalently, a state-space search problem that can be represented by a decision tree or AND/OR graph. It can be solved using standard dynamic programming or using a search algorithm such as AO*.

Because the rover model satisfies the Markov property, it is easy to show that given an arbitrary PRU, an initial resource allocation and a reward function, the optimal policy for the corresponding MDP provides an optimal control strategy for the rover [36].

We note that the number of states of the MDP is bounded by the product of the number of levels, the maximum number of alternative modules per level, the number of discrete quality levels, and the number of possible resource vectors. While resources could vary over a wide range, the size of the control policy can be reduced by using coarse units. Therefore, unit choice introduces a tradeoff between the size of the policy and its effectiveness. An implementation of the policy construction algorithm for problems that involve one resource confirms the intuition that the optimal policy can be approximated with a coarse resource unit [36]. This observation leads to a significant reduction in policy size and construction time.

4.3 Optimal Control of Multiple Activities Using Opportunity Cost

Consider now the control of a complex plan composed of $n + 1$ PRUs. One obvious approach is to generalize the solution for a single PRU to sequences of PRUs. That is, one could construct a large MDP for the combined sequential decision problem including the entire set of $n+1$ PRUs. Each state must include an indicator of the activity (or PRU) number, i, leading to a general state represented as $[i, l, q, r]$, $i \in \{0, 1, \ldots, n\}$. However, our objective is to eliminate the need to solve complex MDPs on-board by the rover. Transmitting to the rover a very large policy for the entire plan is also unacceptable. Instead, we examine a technique to factor the effect of the remaining plan on the current policy using the notion of opportunity cost.

We want to measure the effect of the remaining n PRUs on the execution of the first one. This can be expressed in a way that preserves optimality while suggesting an efficient approach to meta-level control that does not requires run-time construction of the entire policy.

Definition 1. *Let* $V^*(i,r) = V([i,l_0,0,r])$ *for* $i \leq n$, *and* $V^*(n+1,r) = 0$. $V^*(i,r)$ *denotes the expected value of the optimal policy for the last* $n-i$ *PRUs with resources* r.

To compute the optimal policy for the i-th PRU, we can simply use the following modified reward function.

$$U_i'(q,r) = U_i(q) + V^*(i+1,r) \tag{4}$$

In other words, the reward for completing the i-th activity is the sum of the immediate reward and the reward-to-go for the remaining PRUs using the remaining resources. Therefore, the best policy for the first PRU can be calculated if we use the following reward function for final states:

$$U_0'(q,r) = U_0(q) + V^*(1,r) \tag{5}$$

Definition 2. *Let* $OC(r,\Delta r) = V^*(1,r) - V^*(1,r-\Delta r)$ *be the resource* **opportunity cost function**.

The opportunity cost measures the loss of expected value due to reduction of Δr in resource availability when starting to execute the last n PRUs.

Definition 3. *Let the* **OC-policy** *for the first PRU be the policy computed with the following reward function:*

$$U_0'(q,r) = U_0(q) - OC(r_0, r_0 - r)$$

The OC-policy is the policy computed by deducting from the actual reward for the first task the opportunity cost of the resources it consumed.

Theorem 1. *Controlling the first PRU using the OC-policy is globally optimal.*

Proof: From the definition of $OC(r,\Delta r)$ we get:

$$V^*(1, r_0 - \Delta r) = V^*(1,r_0) - OC(r_0, \Delta r) \tag{6}$$

To compute the optimal schedule we need to use the reward function defined in Equation 4 that can be rewritten as follows.

$$U_0'(q, r_0 - \Delta r) = U_0(q) + V^*(1,r_0) - OC(r_0, \Delta r) \tag{7}$$

Or, equivalently:

$$U_0'(q,r) = U_0(q) + V^*(1,r_0) - OC(r_0, r_0 - r) \tag{8}$$

But this reward function is the same as the one used to construct the OC-policy, except for the added constant $V^*(1, r_0)$. Because adding a constant to a reward function does not affect the policy, the optimality of the policy is preserved.

Theorem 1 suggests an optimal approach to control an arbitrary set of $n+1$ activities by first using an OC-policy for the first PRU that takes into account the resource opportunity cost of the remaining n activities. Then, the OC-policy for the second PRU is used taking into account the opportunity cost of the remaining $n-1$ activities and so on.

4.4 Using Estimated Opportunity Cost and Precompiled Policies

How can we exploit the modularity introduced in the previous section to meet the objective of minimizing on-line planning? In particular, we want to avoid any complex procedure that involves computing the exact opportunity cost or re-constructing the corresponding OC-policies on-board. We also want to avoid constructing these policies at the control center because transmitting them to the rover is not feasible. Instead, a solution based on the following two principles has been developed [36].

1. A fast approximation scheme is derived off-line to estimate the opportunity cost of an arbitrary given plan; and
2. Pre-compiled policies are stored on-board to control each activity for different levels of opportunity cost.

We have examined several approaches to estimating the opportunity cost of one resource. Function approximation techniques seem to be suitable for learning the opportunity cost from samples of examples for which we can compute the exact cost off-line. In order to avoid computing a new policy (for a single PRU) each time the opportunity cost is revised, we can divide the space of opportunity cost into a small set of regions representing typical situations. For each region, an optimal policy is computed off-line and stored in a library. At run-time, the system must first estimate the opportunity cost and then use the most appropriate pre-compiled policy from the library. These policies remain valid as long as the overall task structure and the utility function are fixed.

5 Hierarchical Reinforcement-Learning Approach

Another approach to the rover control problem that exploits its MDP representation is based on hierarchical reinforcement learning [2]. There has been increased interest in recent years in classes of MDPs that are naturally decomposable and in developing special-purpose techniques for these classes [4]. The rover control problem can be modeled as a weakly-coupled MDP, which falls in this category. A weakly-coupled MDP is an MDP that has a natural decomposition into a set of subprocesses. The transition from one subprocess to another requires entry into one of a small set of bottleneck states. Because the subprocesses are only connected through a small set of states, they are "almost" independent. The

common intuition is that weakly-coupled MDPs should require less computational effort to solve than arbitrary MDPs.

The algorithm that was investigated is a reinforcement-learning version of a previously studied planning algorithm for weakly-coupled MDPs [13]. The planning algorithm is model-based, whereas the learning algorithm requires only information from experience trajectories and knowledge about which states are the bottleneck states. This can be beneficial for problems where only a simulator or actual experience are available. The algorithm fits into the category of hierarchical reinforcement learning (e.g., [32]) because it learns simultaneously at the state level and at the subprocess level. We note that other researchers have proposed methods for solving weakly-coupled MDPs [16,20,26], but very little work has been done in a reinforcement learning context.

The hierarchical algorithm has been compared with Q-learning; it is shown to perform better initially, but it fails to converge to the optimal policy. Hence, the hierarchical approach could be beneficial in situations in which computation time is limited and a fast approximation of the optimal policy is needed. Surprisingly, a third algorithm which is given the optimal values for the bottleneck states at the start learns more slowly. We discuss this counterintuitive observation at the end of this section.

5.1 Solving MDPs Using Reinforcement Learning

Consider an MDP that contains a finite set S of states, with s_0 being the start state. For each state $s \in S$, A_s is a finite set of possible actions. P is the table of transition probabilities, where $P(s'|s,a)$ is the probability of a transition to state s' given that the agent performed action a in state s. R is the reward function, where $R(s,a)$ is the reward received by the agent given that it chose action a in state s. In this section, we use the infinite-horizon discounted optimality criterion. Formally, the agent should maximize

$$E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(s_t))\right], \tag{9}$$

where $\gamma \in [0,1]$ is the discount factor.

Algorithms for MDPs often solve for *value functions*. For a policy π, the state value function, $V^\pi(s)$, gives the expected total reward starting from state s and executing π. The state-action value function, $Q^\pi(s,a)$, gives the expected total reward starting from state s, executing action a, and executing π from then on.

Reinforcement learning techniques are particularly useful when only a a simulator or real experience are available [31]. With these techniques, experience trajectories are used to learn a value function for a good policy. Actions taken on a trajectory are usually greedy with respect to the current value function, but *exploratory* actions must also be taken in order to discover better policies. One widely-used reinforcement learning algorithm is Q-learning [33], which updates the state-action value function after each transition from s to s' under action a

with the following rule:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[R(s,a) + \gamma \max_{a'} Q(s',a') - Q(s,a) \right], \tag{10}$$

where α is the learning rate.

5.2 Reinforcement Learning for Weakly-Coupled MDPs

Consider an MDP with a state set S that is partitioned into disjoint subsets $S_1, \ldots, S_m$. The *out-space* of a subset S_i, denoted $O(S_i)$, is defined to be the set of states not in S_i that are reachable in one step from some state in S_i. The set of states $B = O(S_1) \cup \cdots \cup O(S_m)$ that belong to the out-space of at least one subset comprise the set of *bottleneck* states. If the set of bottleneck states is relatively small, we call the MDP *weakly-coupled*.

In [13], the authors describe an algorithm for weakly-coupled MDPs that can be described as a type of policy iteration. Initially, values for the bottleneck states are set arbitrarily. The low-level policy improvement phase involves solving each subproblem, treating the bottleneck state values as terminal rewards. The high-level policy evaluation phase consists of reevaluating the bottleneck states for these policies. Repeating these phases guarantees convergence to the optimal policy in a finite number of iterations.

The rules for backpropagating value information in the reinforcement learning algorithm are derived from the two phases mentioned above. Two benefits of this approach are that it does not require an explicit model and that learning can proceed simultaneously at the high level and at the low level.

Two different value functions must be maintained: a low-level state-action value function Q defined over all state-action pairs and a high-level state value function V_h defined only over bottleneck states. The low-level part of the learning is described as follows. Upon a transition to a non-bottleneck state, the standard Q-learning backup is applied. However, when a bottleneck state $s' \in B$ is encountered, the following backup rule is used:

$$Q(s,a) \leftarrow Q(s,a) + \alpha_l \left[R(s,a) + \gamma V_h(s') - Q(s,a) \right], \tag{11}$$

where α_l is a learning rate. For the purposes of learning, the bottleneck state is treated as a terminal state, and its value is the terminal reward. High-level backups occur only upon a transition to a bottleneck state. The backup rule is:

$$V_h(s) \leftarrow V_h(s) + \alpha_h [R + \gamma^k V_h(s') - V_h(s)], \tag{12}$$

where k denotes the number of time steps elapsed between the two bottleneck states, R is the cumulative discounted reward obtained over this time, and α_h is a learning rate.

It is possible to alternate between phases of low-level and high-level backups or to perform the backups simultaneously. Whether either approach converges to an optimal policy is an open problem. We chose the latter for our experiments because our preliminary work showed it to be more promising.

5.3 The Rover Model

The rover control problem fits nicely the weakly-coupled MDP framework. In this section we evaluate the approach using a simple scenario. In this scenario, a rover is to operate autonomously for a period of time. As in Section 4, the overall plan is composed of a sequence of activities, each of which includes a set of data gathering actions with respect to a certain target. Each activity has an associated priority and estimated difficulty of obtaining the data. The rover must make decisions about which activities to perform and when to move from one target to the next. The goal is to maximize the value of the collected data over a given time period.

The action set consists of taking a picture, performing a spectrometer experiment, and traversing to the next target in the sequence. Spectrometer experiments take more time and are less predictable than taking pictures, but they yield better data. The time to traverse between targets is a noisy function of the distance between them. The state features are the remaining time, the current target number (from which priority and estimated difficulty are implicitly determined), the number of pictures taken of the current target, and whether or not satisfactory spectrometer data has been obtained. Formally, $S = T \times I \times P \times E$, where $T = \{0 \text{ min}, 5 \text{ min}, \ldots, 300 \text{ min}\}$ is the set of time values; $I = \{1, 2, 3, 4, 5\}$ is the set of targets; $P = \{0, 1, 2\}$ is the set of values for pictures taken; and $E = \{0, 1\}$ is the set of values for the quality of the spectrometer data. The start state is $s_0 = \langle 300, 1, 0, 0 \rangle$. The sequence of targets used for our experiments is shown in Table 1.

Table 1. The sequence of targets for the rover to investigate

Target	Priority	Estimated difficulty	Distance to next target
1	8	medium	3 m
2	5	hard	5 m
3	3	easy	7 m
4	2	easy	3 m
5	9	hard	N/A

A nonzero reward can only be obtained upon departure from a target location and is a function of the priority of the target and the data obtained about the target. The task is episodic with $\gamma = 1$. An episode ends when the time component reaches zero or the rover finishes investigating the last target. The aim is to find a policy that maximizes the expected total reward across all targets investigated during an episode.

In order to see how this problem fits into the weakly-coupled MDP framework, consider the set of states resulting from a traversal between targets. In all of these states, the picture and spectrometer components of the state are reset to zero. The set $B = T \times I \times \{0\} \times \{0\}$ is taken to be the set of bottleneck states,

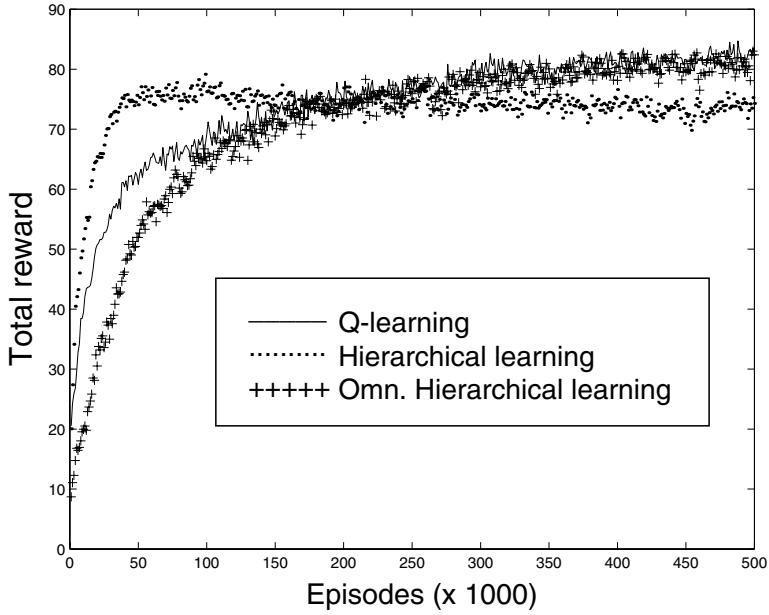


Fig. 4. Learning curves for Q-learning, hierarchical learning, and omniscient hierarchical learning

and it is over this set that we define the high-level value function. Note that the bottleneck states comprise only 300 of the problem's 1,800 states.

5.4 Experiments

The hierarchical algorithm has been tested against Q-learning using the above scenario. In addition, we tested an algorithm that we call the *omniscient* hierarchical learning algorithm. This algorithm is the same as the hierarchical algorithm, except that the values for the bottleneck states are fixed to optimal from the start, and only low-level backups are performed. By fixing the bottleneck values, the problem is completely decomposed from the start. Of course, this cannot be done in practice, but it is interesting for the purpose of comparison.

For the experiments, all values were initialized to zero, and we used ϵ-greedy exploration with $\epsilon = 0.1$ [31]. For the results shown, all of the learning rates were set to 0.1 (we obtained qualitatively similar results with learning rates of 0.01, 0.05, and 0.2). Figure 4 shows the total reward per episode plotted against the number of episodes of learning. The points on the curves represent averages over periods of 1000 episodes.

A somewhat counterintuitive result is that the omniscient hierarchical algorithm performs worse than both the original hierarchical algorithm and Q-learning during the early stages. One factor contributing to this is the initialization of the state-action values to zero. During the early episodes of learning, the

value of the "leave" action grows more quickly than the values for the other actions because it is the only one that leads directly to a highly-valued bottleneck state. Thus the agent frequently leaves a target without having gathered any data. This result demonstrates that decomposability does not always guarantee a more efficient solution.

The second result to note is that the hierarchical algorithm performs better than Q-learning initially, but then fails to converge to the optimal policy. It is intuitively plausible that the hierarchical algorithm should go faster, since it implicitly forms an abstract process involving bottleneck states and propagates value information over multiple time steps. It also makes sense that the algorithm does not converge once we consider that the high-level backups are *off policy*. This means that bottleneck states are evaluated for the policy that is being executed, and this policy always includes non-greedy exploratory actions. Algorithms such as Q-learning, on the other hand, learn about the policy that is greedy with respect to the value function regardless of which policy is actually being executed.

6 Discussion

We have described two decision-theoretic approaches to control planetary rovers. The two approaches share several characteristics: they both use an MDP representation of the control problem and they both exploit the fact that the plan components are only loosely-coupled. Both techniques use approximations; in previous experimentation with small scale problem instances, both produced near-optimal control. However, it is hard to predict how the optimality of the resulting control policies degrades with problem complexity and which one of the techniques will be more robust. This remains an open problem.

Both of the techniques are designed to minimize the complexity of on-line planning, and they both rely on pre-computing and storing control policies on-board. The size of these control policies could be substantial, but the required space is significantly smaller than the space needed for a complete policy for the entire plan. One advantage of the adaptive planning approach is that the control policies can be applied to an *arbitrary* plan, as long as the plan components are defined using known PRUs. This is not the case with the hierarchical reinforcement-learning technique. However, the latter has the advantage of not relying on knowing the exact model of the environment. Moreover, learning could be used on-board to refine a pre-calculated policy and adapt it to the real environment of operation.

A considerable amount of work remains to be done to examine the scalability of both solution techniques. In theory, if reinforcement learning is applied correctly, it can handle very large problems. However, in practice, this is a challenging problem. The scalability of the adaptive planning approach may be more predictable. We expect it to handle well larger plans in terms of the number of activities and their complexity. However, estimating the opportunity cost of multiple resources seems hard. The quality of these estimates degrades as the number

of possible activities grows. Resource costs, unfortunately, are not independent, leading to exponential growth in the number of necessary pre-compiled policies as the number of resources increases.

Another important source of complexity comes from generalizing the topology of the plan, allowing cycles within an activity and partially-ordered activities in a plan. The utility of repeating an activity, such as taking pictures or collecting samples, is non-additive over the set of repetitions and may depend on the degree of success with previous attempts. This could lead to a significant increase in the number of state variables of the MDP. Constructing policies for MDPs with cycles is harder, but this has been addressed effectively by existing dynamic-programming and reinforcement-learning algorithms.

Introducing temporal and other constraints on activities is another important generalization that is the focus of current research. It would allow us to represent scientific or operational constraints on rover operations, such as illumination for imaging (or lack thereof for some spectral measurements), temperature for instrument performance, or pre-defined communication windows with Earth or orbiting relay satellites. Such constraints introduce interaction between plan components that we managed to avoid so far.

Yet another source of complication is partial observability of the quality of scientific data. If quality represents a simple aspect of the collected data, such as the resolution of an image, then we can assume that quality is fully observable. However, if we want to measure the actual quality of the scientific data, this can lead to additional tradeoffs in planning and execution. Estimating quality may be a non-trivial computational task that returns imperfect information. Integrating on-board data interpretation processes as part of the overall planning and control problem is another focus of current research efforts.

The results surveyed in this paper are part of a long term research program to make planetary rovers more autonomous and more productive. It is important to maintain in this effort a delicate balance between the various objectives and to aim for an appropriate level of autonomy for the task. Given the necessary interactions of the scientists with "their" rover on a planetary surface (to monitor the exploration and make ultimate determination of what is interesting), it does not seem necessary to give rovers a single goal, such as "find life" and leave it to its own devices. As rovers become more versatile and scientists produce more sophisticated science interpretation instruments and on-board analysis programs, we can imagine the level of autonomy increasing, requiring more sophisticated on-board planning and execution mechanisms. For the time being, the approaches we described match the level of autonomy needed to perform the mission efficiently and effectively.

References

1. Bapna, D., Rollins, E., Murphy, J., Maimone, E., Whittaker, W., and Wettergreen, D.: The Atacama Desert Trek: Outcomes. *IEEE International Conference on Robotics and Automation (ICRA-98)* (1998) 597–604
2. Bernstein, D.S., and Zilberstein, S.: Reinforcement Learning for Weakly-Coupled MDPs and an Application to Planetary Rover Control. *European Conference on Planning* (2001)
3. Bernstein, D.S., Zilberstein, S., Washington, R., and Bresina, J.L.: Planetary Rover Control as a Markov Decision Process. *Sixth International Symposium on Artificial Intelligence, Robotics, and Automation in Space* (2001)
4. Boutilier, C., Dean, T., and Hanks, S.: Decision-Theoretic Planning: Structural Assumptions and Computational Leverage. *Journal of Artificial Intelligence Research* **1** (1999) 1–93
5. Boyan, J.A., and Littman, M.L.: Exact Solutions to Time-Dependent MDPs. *Advances in Neural Information Processing Systems* MIT Press, Cambridge, MA (2001)
6. Bresina, J.L., Golden, K., Smith, D.E., and Washington, R.: Increased Flexibility and Robustness of Mars Rovers. *Fifth International Symposium on Artificial Intelligence, Robotics and Automation in Space* (1999)
7. Bresina, J.L., and Washington, R.: Expected Utility Distributions for Flexible, Contingent Execution. *AAAI-2000 Workshop: Representation Issues for Real-World Planning Systems* (2000)
8. Bresina, J.L., and Washington, R.: Robustness Via Run-Time Adaptation of Contingent Plans. *AAAI Spring Symposium on Robust Autonomy* (2001)
9. Bresina, J.L., Bualat, M., Fair, M., Washington, R., and Wright, A.: The K9 On-Board Rover Architecture. *European Space Agency (ESA) Workshop on On-Board Autonomy* (2001)
10. Cardon, S., Mouaddib, A.-I., Zilberstein, S., and Washington, R.: Adaptive Control of Acyclic Progressive Processing Task Structures. *Seventeenth International Joint Conference on Artificial Intelligence* (2001) 701–706
11. Christian, D., Wettergreen, D., Bualat, M., Schwehr, K., Tucker, D., and Zbinden, E.: Field Experiments with the Ames Marsokhod Rover. *Field and Service Robotics Conference* (1997)
12. Dean, T., and Boddy, M.: An Analysis of Time-Dependent Planning. *Seventh National Conference on Artificial Intelligence* (1988) 49–54
13. Dean, T., and Lin, S.-H.: Decomposition Techniques for Planning in Stochastic Domains. *Fourteenth International Joint Conference on Artificial Intelligence* (1995) 1121–1127
14. Estlin, T., Gray, A., Mann, T., Rabideau, G., Castano, R., Chien, S., and Mjolsness, E.: An Integrated System for Multi-Rover Scientific Exploration. *Sixteenth National Conference on Artificial Intelligence* (1999) 541–548
15. Fikes, R., and Nilsson, N.: Strips: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence* **2** (1971) 189–208
16. Foreister, J.-P., and Varaiya, P.: Multilayer Control of Large Markov Chains. *IEEE Transactions on Automatic Control* **23**(2) (1978) 298–304
17. Fukunaga, A., Rabideau, G., Chien, S., Yan, D.: Toward an Application Framework for Automated Planning and Scheduling. *International Symposium on Artificial Intelligence, Robotics and Automation for Space* (1997)

18. Green, C.: Application of Theorem Proving to Problem Solving. *First International Joint Conference on Artificial Intelligence* (1969) 219–239
19. Hansen, E.A., and Zilberstein, S.: LAO*: A Heuristic Search Algorithm that Finds Solutions with Loops. *Artificial Intelligence* **129**(1-2) (2001) 35–62
20. Hauskrecht, M., Meuleau, N., Kaelbling, L.P., Dean, T., and Boutilier, C.: Hierarchical Solution of Markov Decision Processes Using Macro-Actions. *Fourteenth International Conference on Uncertainty in Artificial Intelligence* (1998)
21. Kaelbling, L., Littman, M., and Cassandra, A.: Planning and Acting in Partially Observable Stochastic Domains. *Artificial Intelligence* **101**(1-2) (1998) 99–134
22. Limonadi, D.: Smart Lander Reference Surface Scenario and Reference Vehicle Description. Jet Propulsion Laboratory Interoffice Memorandum, November 2 (2001)
23. Mishkin, A.H., Morrison, J.C., Nguyen, T.T., Stone, H.W., Cooper, B.K., and Wilcox, B.H.: Experiences with Operations and Autonomy of the Mars Pathfinder Microrover. *IEEE Aerospace Conference* (1998)
24. Mouaddib, A.-I., and Zilberstein, S.: Optimal Scheduling of Dynamic Progressive Processing. *Thirteenth Biennial European Conference on Artificial Intelligence* (1998) 449–503
25. Muscettola, N., Nayak, P.P., Pell, B., and Williams, B.C.: Remote Agent: To Boldly Go Where No AI System Has Gone Before. *Artificial Intelligence* **103**(1-2) (1998) 5–47
26. Parr, R.: Flexible Decomposition Algorithms for Weakly-Coupled Markov Decision Problems. *Fourteenth International Conference on Uncertainty in Artificial Intelligence* (1998)
27. Puterman, M.L.: Markov Decision Processes – Discrete Stochastic Dynamic Programming. John Wiley & Sons, Inc., New York, NY (1994)
28. Sacerdoti, E.D.: Planning in a Hierarchy of Abstraction Spaces. *Artificial Intelligence* **5**(2) (1974) 115–135
29. Simmons, R., and Koenig, S.: Probabilistic Robot Navigation in Partially Observable Environments. *Fourteenth International Joint Conference on Artificial Intelligence* (1995) 1080–1087
30. Stoker, C., Roush, T., and Cabrol, N.: Personal communication (2001)
31. Sutton, R.S., and Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge, MA (1998)
32. Sutton, R.S., Precup, D., and Singh, S.: Between MDPs and Semi-MDPs: Learning, Planning, and Representing Knowledge at Multiple Temporal Scales. *Artificial Intelligence*, **112** (2000) 181–211
33. Watkins, C.: Learning from Delayed Rewards. PhD Thesis, Cambridge University, Cambridge, England (1989)
34. Wellman, M.P., Larson, K., Ford, M., and Wurman, P.R.: Path Planning under Time-Dependent Uncertainty. *Eleventh Conference on Uncertainty in Artificial Intelligence* (1995) 532–539.
35. Zilberstein, S., and Mouaddib, A.-I.: Reactive Control of Dynamic Progressive Processing. *Sixteenth International Joint Conference on Artificial Intelligence* (1999) 1268–1273
36. Zilberstein, S., and Mouaddib, A.-I.: Adaptive Planning and Scheduling of On-Board Scientific Experiments. *European Space Agency (ESA) Workshop on On-Board Autonomy* (2001)

Author Index

Alami, Rachid 1

Bach, Joscha 71
Beetz, Michael 21, 36
Berger, Ralf 71
Bernstein, Daniel S. 270
Bibel, Wolfgang 193
Bothelho, Silvia Silva da Costa 1
Brunswieck, Birger 71
Buck, Sebastian 36
Burgard, Wolfram 52
Burkhard, Hans-Dieter 71

Coradeschi, Silvia 89

Ghallab, Malik 157
Gini, Maria 211
Gollin, Michael 71

Hertzberg, Joachim 249
Hofhauser, Andreas 21
Hougen, Dean 211

Kabanza, Froduald 123
Karlsson, Lars 106
Koenig, Sven 140

Lamine, Khaled Ben 123

Likhachev, Maxim 140

McCarthy, Colleen E. 179
Moors, Mark 52
Morisset, Benoit 157
Mouaddib, Abdel-Illah 270

Papanikolopoulos, Nikolaos 211
Pollack, Martha E. 179
Pratihar, Dilip Kumar 193

Ramakrishnan, Sailesh 179
Rybski, Paul 211

Saffiotti, Alessandro 89
Sandewall, Erik 226
Schiavinotto, Tommaso 106
Schmitt, Thorsten 36
Schneider, Frank 52
Schönherr, Frank 249
Stoeter, Sascha 211

Tsamardinos, Ioannis 179

Washington, Richard 270

Zilberstein, Shlomo 270