Luiz Chaimowicz

Coordenação Dinâmica de Robôs Cooperativos: Uma Abordagem Utilizando Sistemas Híbridos

Tese apresentada ao Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Doutor em Ciência da Computação.

Universidade Federal de Minas Gerais

Belo Horizonte, Junho de 2002

Luiz Chaimowicz

Dynamic Coordination of Cooperative Robots: A Hybrid Systems Approach

Thesis presented to the Doctoral Program in Computer Science of the Federal University of Minas Gerais in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Universidade Federal de Minas Gerais

Belo Horizonte, June, 2002

Resumo

Esta tese aborda o problema da modelagem e coordenação de múltiplos robôs na execução de tarefas cooperativas. É proposta e apresentada uma nova metodologia que usa alocação dinâmica de papéis na coordenação de múltiplos robôs. Basicamente, cada robô executa um papel que define as suas ações durante a cooperação. Dinamicamente assumindo e trocando papéis de forma sincronizada, os robôs são capazes de executar tarefas cooperativas com sucesso, se adaptando a eventos inesperados que possam ocorrer no ambiente e obtendo um melhor desempenho. As tarefas cooperativas e a alocação dinâmica de papéis são modeladas utilizando-se sistemas híbridos. Um sistema híbrido pode ser visto como uma composição de estados discretos e equações contínuas, sendo bastante adequado para a representação de sistemas com múltiplos robôs. Um autômato híbrido é utilizado para representar o comportamento de cada robô na execução da tarefa e a composição de vários autômatos permite que a tarefa cooperativa como um todo seja modelada. Isso permite uma melhor representação e formalização da execução de tarefas cooperativas e provê um arcabouço para obtenção de resultados formais sobre a cooperação. Vários experimentos são realizados em diversas tarefas cooperativas, utilizando-se tanto robôs reais como simulados. Para isso, foi desenvolvido um simulador chamado de MuRoS, que permite a simulação de múltiplos robôs na execução de tarefas cooperativas. Os experimentos demonstram a efetividade da metodologia proposta na modelagem e execução de tarefas cooperativas.

Abstract

This thesis addresses the problem of how to model and coordinate multiple robots in the execution of cooperative tasks. We present a novel methodology that uses a dynamic role assignment mechanism for coordinating multiple robots. Basically, each robot in the team performs a role that determines its actions during the cooperation. Dynamically assuming and exchanging roles in a synchronized manner, the robots are able to perform the task successfully, having a better performance and adapting to unexpected events in the environment. We use a hybrid systems framework to model the execution of cooperative tasks and the role assignment mechanism. Hybrid systems can be viewed as a composition of discrete states and continuous equations, being adequate to represent multi-robot systems. We use a hybrid automaton to specify the behavior of each robot and the parallel composition of automata to model the cooperative task as a whole. This allows us to better describe and formalize the execution of cooperative tasks by multi-robot teams and provides a framework for obtaining formal results about the cooperation. Experiments with the role assignment mechanism and the hybrid systems modeling are performed in different tasks, both in simulated and real environments. A multi-robot simulator called MuRoS has been developed in order to simulate the execution of cooperative tasks. The experiments demonstrate the effectiveness of the proposed methodology in the modeling and execution of cooperative multi-robot tasks.

Agradecimentos

Eu gostaria de agradecer a todos que de alguma forma contribuíram para o desenvolvimento deste trabalho.

Ao meu orientador, professor Mário F. M. Campos, pela orientação e amizade durante esses 4 anos, e pelo aprendizado de que mais importante do que este resultado final é todo o processo pelo qual se passa para chegar até aqui.

Ao meu co-orientador, professor Vijay Kumar, que me recebeu tão bem no GRASP Laboratory, na Universidade da Pennsylvania, e cuja contribuição foi fundamental para o alto nível desta tese.

Aos membros da minha banca, professores Alberto Elfes, Antônio Alfredo Loureiro, Claudionor Coelho e Marcel Bergerman, pelas sugestões para a melhoria deste trabalho.

Aos amigos e colegas do VERLAB e do GRASP, pela excelente atmosfera de trabalho e valiosa troca de idéias durante o desenvolvimento desta tese. Em especial ao Denilson Laudares, companheiro de todas as horas durante esse longo doutorado, ao Guilherme Pereira, Bruno Pimentel, Rafael Fierro, Tom Sugar e Peng Song.

Aos amigos Ado Jório e Rodrigo Carceroni, pela contínua motivação em se seguir uma carreira científica.

Aos colegas, professores e funcionários do Departamento de Ciência da Computação da UFMG. Em especial ao professor José Nagib C. Árabe, meu orientador de iniciação científica e mestrado, e à Renata Viana, pela extrema presteza e boa vontade para resolver todos os problemas burocráticos envolvidos em uma tese de doutorado.

À CAPES e CNPq pelo suporte financeiro.

Aos meus pais e minha família, pelo apoio incondicional não só durante esse trabalho mas durante toda a minha vida.

À Luciana, minha querida esposa, que esteve sempre ao meu lado dando toda força e apoio que eu precisava para poder continuar essa jornada. Este trabalho não seria possível sem você.

Acknowledgments

I would like to thank all the people that, in some way, contributed for the development of this work. A special thanks to:

My advisor, Professor Mário F. M. Campos, for the guidance and friendship during these four years and for the lesson that, more important than this final result, is the whole process that we have to pass through to get here.

My co-advisor, Professor Vijay Kumar, who kindly received me in the GRASP Laboratory at University of Pennsylvania and significantly contributed for the high quality of this thesis.

The members of my committee, professors Alberto Elfes, Antônio Alfredo Loureiro, Claudionor Coelho, and Marcel Bergerman, for their suggestions to the improvement of this work.

The colleagues and friends at VERLAB and GRASP, for the excellent work atmosphere and the valuable discussions during the development of this thesis. In special, I would like to acknowledge the support and collaboration of Denilson Laudares, Guilherme Pereira, Bruno Pimentel, Rafael Fierro, Tom Sugar and Peng Song.

My friends Ado Jório and Rodrigo Carceroni, for the continuous motivation to pursue a research career.

The colleagues, professors and staff of the Computer Science Department of UFMG. In special, Professor José Nagib C. Árabe, my former advisor, and Renata Viana, for the help with all the paperwork related to a doctoral thesis.

CAPES and CNPq for the financial support.

My parents and my family, for the unconditional support not only during this work but in all my life.

Luciana, my beloved wife, who has always been on my side, giving me all the strength and support necessary to continue this journey. This work would not be possible without you.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

Cooperative robotics has been an active research field in the last few years. Fundamentally, it consists of a group of robots working cooperatively to execute various types of tasks in order to increase the robustness and efficiency of task execution. The use of multi-robot teams brings several advantages over single robot approaches. Firstly, depending on the type of the task, multiple robots can execute it more efficiently by dividing the work among the team. More than that, groups of simpler and less expensive robots working cooperatively can be used instead of an expensive specialized robot. Robustness is also increased in certain tasks by having robots with redundant capabilities and dynamically reconfiguring the team in case of robot failures.

Additionally, there are certain kinds of tasks, called tightly coupled tasks, that cannot be executed by a single robot and require the use of multiple robots working cooperatively. To execute tightly coupled tasks, robots must act in a highly coordinated fashion and this normally requires some knowledge about the states and actions of the teammates, either through implicit (sensory perception) or explicit communication. Further, in many cases, each robot is critical to the task and an individual failure could cause a failure of the task as a whole. These are different from loosely coupled tasks, that can be executed by a single

robot alone, but, in general, have better performance when a team of robots is used. When executing loosely coupled tasks, robots can act independently from each other and strict coordination is not required. A classical example of a loosely coupled task is foraging, in which a group of robots must search an area for certain items and individually retrieve these items to a certain location. On the other hand, a typical example of a tightly coupled task is the cooperative manipulation, in which two or more robots transport large objects between different positions in the environment.

The research in cooperative robotics has a strong biological inspiration. Several animal societies depend heavily on cooperative work in order to execute tasks that could not be done by individual members or that are better performed by multiple agents, such as to gather food, protect the group, etc. Some examples of this cooperation are the foraging behavior of ants and bees, flight formations of birds and migrations of herds of elephants and buffalos. Humans also use cooperation in the execution of their tasks. For example, the construction of a bridge requires the cooperative work of several specialized workers, from engineers to contractors, and it is also necessary a high level of coordination in order to avoid failures and to meet deadlines. Even simpler human tasks like carrying a heavy load or cleaning a house benefit from cooperative work and require different levels of coordination.

In the same way as humans, robots must be coordinated in order to execute cooperative work. The coordination of multi-robot teams in dynamic environments is a very challenging task. Basically, the actions performed by each team member during each phase of the cooperation must be specified considering several aspects such as robot properties, task requirements, information flow, and characteristics of the environment. Thus, different levels of coordination are required by different tasks and robots. Also, the coordination mechanism should help multi-robot teams in the synchronization of their actions and in the exchange of information necessary for the task execution. Besides that, it should provide flexibility and adaptability, allowing the robots to complete cooperative tasks more efficiently and robustly.

A precise modeling of multi-robot cooperation is also a very important aspect of co-

operative robotics. Cooperative robotics systems normally require the representation of both continuous and discrete dynamics, together with synchronization, communication, etc. The modeling technique should address all these requirements and also provide ways of obtaining some formal results about the cooperative system. All these factors make the development of adequate coordination mechanisms and the modeling of cooperative tasks some of the key issues in cooperative robotics.

1.2 Problem Definition and Objectives

The problem that we are addressing in this thesis can be stated as follows: given a group of robots, with homogeneous or heterogeneous capabilities, how to **model** and **coordinate** multi-robot cooperation so that the robots will be able to perform different types of cooperative tasks in dynamic environments.

Thus, the main objective of this thesis is to develop a mechanism that allows the coordination of multiple robots in different tasks, ranging from loosely coupled to tightly coupled tasks. This mechanism should be built using a formal theory that allows a precise modeling of the cooperation, including the interactions among the robots, the exchange of information, and task synchronization.

1.3 Contributions

To accomplish these objectives, we propose a dynamic role assignment mechanism, in which robots can dynamically change their behavior being able to successfully execute cooperative tasks. We consider that a role is a function that one or more robots perform during the execution of a cooperative task. It depends on the internal robot state and on information about the environment and other robots, and defines the variables, information flow, and the set of controllers that will be controlling the robot in each moment. By dynamically assuming and exchanging roles in a synchronized manner, the robots are able to perform cooperative tasks, adapting to unexpected events in the environment and improving their

individual performance in benefit of the team.

We model cooperative robotics and the role assignment mechanism using a hybrid systems framework. Hybrid systems can be viewed as a composition of discrete states and continuous dynamics, being adequate to represent multi-robot systems. Using a hybrid automaton, we are able to represent roles, role assignments, continuous controllers, and discrete variables related to each robot. The composition of these automata allows us to model the execution of cooperative tasks.

Experiments with the role assignment mechanism on different cooperative tasks have been performed in simulations and in real robots, showing the effectiveness of the proposed methodology. A multi-robot simulator called MuRoS has been developed in order to simulate the execution of cooperative tasks. Implemented using object orientation in the MS Windows environment, MuRoS has allowed the study of different aspects of cooperative robotics in several application domains.

In summary, the main contributions of this thesis are:

1. The development of a novel mechanism for the coordination of multiple robots in the execution of cooperative tasks. As mentioned, this mechanism uses dynamic role assignment in order to allocate actions and dynamically coordinate multi-robot teams in the performance of different applications.

2. The use of a hybrid systems framework in order to model the cooperative execution of tasks by multiple robots, allowing a better formalization of the cooperation. Using this framework, we are able to describe and study important aspects of cooperative systems such as information exchange and synchronization.

Another important contribution of this thesis, although it is not a primary contribution, is the implementation of a multi-robot simulator (MuRoS) for the execution of cooperative tasks. We have used it to study several aspects of cooperative robotics and our intention is to make it available to other research groups and use it in different applications.

1.4 Organization of this work

This work is organized as follows: in the next chapter we discuss some related work, focusing on coordination mechanisms for cooperative robotics, cooperative manipulation, and modeling. In Chapter 3, we present the role assignment mechanism and discuss some important aspects of multiple robot coordination such as communication, synchronization, and hierarchy. Chapter 4 gives a brief introduction about hybrid systems and shows our modeling of cooperative robotics using the composition of hybrid automata. In Chapter 5, we present some simulations performed to show the effectiveness of the proposed framework. Basically, we give an overview of the simulator developed for cooperative robotics and then show the simulations of two cooperative tasks, namely cooperative manipulation and cooperative search and rescue. Chapter 6 presents some experimental results of the role assignment mechanism and the hybrid systems modeling in a cooperative manipulation task, in which teams of two and three robots coordinate themselves to transport a large box in an environment containing obstacles. Finally, Chapter 7 brings the conclusion of this thesis and the possibilities for future work.

Chapter 2

Background

The field of cooperative robotics started in the late 80's and early 90's with two seminal cooperative systems: CEBOT [Fukuda and Nakagawa, 1987] and ACTRESS [Asama et al., 1989]. Since then, this field has grown substantially with the development of several multi-robot systems and cooperative applications. It this chapter, we discuss some related work focusing on three aspects of cooperative robotics directly related to this thesis, namely: coordination of multi-robot teams, cooperative manipulation, and formal modeling of robotic tasks. More general surveys of cooperative robotics can be found in [Parker, 2000], [Cao et al., 1997], and [Dudek et al., 1996].

2.1 Coordination of Multi-Robot Teams

Given a group of robots and a task to be performed, how to coordinate the robots in order to successfully complete the task? This is one of the fundamental problems in cooperative robotics and, depending on the type of the task and the robots' characteristics, has different levels of difficulty. For example, coordinating a loosely coupled task with homogeneous robots is much easier than executing a tightly coupled task with a heterogeneous team.

The Behavior-Based Paradigm [Brooks, 1986, Arkin, 1998] is one of the most used approaches to control and coordinate multiple robots in the execution of cooperative tasks.

The behavior-based approach advocates that robots may not need models of the world, and a set of reactive controllers composed by sensing/actuation pairs (called primitive behaviors) are sufficient to control mobile robots in dynamic environments. More elaborated behaviors "emerge" from the composition of low level behaviors allowing robots to perform complex tasks. The main criticisms of behavior-based approaches are that it is difficult to obtain formal results about its correct execution. Sometimes it is difficult to predict what behavior will emerge from the composition of low level behaviors and this lack of modeling can make the analysis and implementation of more complex applications a difficult task. In spite of that, the behavior-based paradigm has been used successfully in various approaches. In [Mataric, 1994], there is an implementation of several basic social behaviors that can be combined to generate more complex social behaviors. These behaviors are used to coordinate multiple robots both in simulated and real environments. Another interesting approach is Alliance [Parker, 1998], a behavior-based software architecture for heterogeneous multi-robot cooperation. It has a fault tolerance mechanism that allows the robots to detect failures in one of the teammates and adapt their behaviors to complete the task. But more traditional deliberative approaches, composed by hierarchical levels, have also been used for controlling cooperative robots. In [Noreils, 1993], for example, there is a description of a leader-follower architecture that is used in cooperative tasks. The architecture is decomposed in three levels (planner, control, and functional), each one with its proper competencies.

An important issue in multi-robot coordination is related to the kind and amount of information that should be exchanged among the robots. Basically, the questions are: does one robot need to have information about the position, state and goals of the others? If so, is this achieved through explicit communication or through observation? Normally, the answer depends on the task being performed. In [Balch and Arkin, 1995], there is a study of explicit versus implicit communication in three different cooperative tasks (forage, consume, and graze). The main conclusion is that explicit communication improves the performance in tasks where the possibility of implicit communication through the environment is small. Similar conclusions can be found in [Parker, 1995], where robot awareness

and action recognition were studied in a "puck moving" mission. An interesting cooperative framework that does not use communication is presented in [Alur et al., 2001]. Equipped only with omnidirectional cameras, a group of robots is capable of executing different tasks such as formation control, mapping and manipulation. Using Kalman filters, the robots can estimate the states of their teammates using visual information, reducing the need for explicit communication. Although these works have shown that it is possible to coordinate multiple robots in certain tasks without using explicit communication, the development of cheaper and more reliable communication mechanisms has motivated the use of explicit communication in multi-robot coordination. In spite of some drawbacks such as poor scalability and lack of reliability in noisy environments, explicit communication has contributed for the development of more general coordination mechanisms that can be used to control multi-robot teams in different types of cooperative tasks.

Another important point in cooperative architectures is the task allocation problem, *i.e.*, how to divide the work among the robots, allocating subtasks to each team member. In most of the cases, the task allocation should be performed dynamically, in such a way that the robots will be able to coordinate themselves with flexibility and adaptability, being capable of successfully completing cooperative tasks. Several researchers have studied the task allocation problem, both for multi-agent software systems and distributed robots. In this work we do not focus on software agents, but some approaches can be found in [Tambe, 1997] and [Shehory and Kraus, 1998] among others. In the cooperative robotics field, one important work is the Alliance architecture [Parker, 1998] mentioned above. Another behavior-based approach for task allocation is presented in [Werger and Mataric, 2000], in which robots broadcast messages with their eligibility in order to coordinate their actions in a multi-target observation task. Other task allocation mechanism is Murdoch [Gerkey and Mataric, 2001], that uses a publish/subscribe architecture. In this system robots broadcast messages with their resources and needs and dynamically allocate tasks based on this information exchange. Another approach is proposed by [Jennings and Kirkwood-Watts, 1998]: the Method of Dynamic Teams is a programming model which addresses the mapping of tasks into dynamic teams of agents.

These teams can grow, shrink, and change dynamically, being programmed according to the task being executed. Task allocation is also studied in [Uchibe et al., 2001], where a method to resolve conflicts between task modules is proposed.

As will be explained in the next chapter, we use a dynamic role assignment mechanism to allocate tasks and control robots in a cooperative task. The term dynamic role assignment has been used in other works such as [Stone and Veloso, 1999], [Castelpietra et al., 2000], and [Emery et al., 2002], all restricted to the multi-robot soccer domain. In [Stone and Veloso, 1999], a role is defined as the specification of an agent's internal and external behaviors. A formation is a set of roles, decomposing the task space. Homogeneous agents can switch roles within formations dynamically, according to predefined triggers that are evaluated at run-time. Each agent knows the current formation and keeps mappings from teammates to roles in the current formation. The agents can communicate using low bandwidth channels and periodically synchronize in a full-communication setting. This mechanism was implemented in the domain of simulated robotic soccer, showing interesting results. The work of [Castelpietra et al., 2000] also implemented a role assignment mechanism in a robot soccer domain, but using a team of three heterogeneous robots. Examples of roles in this domain are: attacker, defender, etc. The coordination protocol uses the internal state of each robot and explicit messages in order to define the team formation and assign roles. In [Emery et al., 2002] the roles are basically the same, but special care is taken during the role assignment to synchronize the robots avoiding possible deadlocks, that in this case is to have two robots with the same role.

2.2 Cooperative Manipulation

Cooperative manipulation is one of the main testbeds for cooperative robotics. In this task, two or more robots transport, reorient, or simply manipulate an object in cooperation. It is a classical example of a tightly coupled task. An interesting characteristic of cooperative manipulation is that the robots are physically coupled through the object. So, in spite of the need for tight coordination, explicit communication can sometimes be

replaced by sensing in some systems. This is discussed in [Donald et al., 1997], where the type and amount of local and global information a team of robots needs to perform a cooperative manipulation task was studied. For example, how much internal state a robot should retain and how much communication is necessary. They perform this analysis using the information invariants methodology [Donald, 1995] and a general conclusion is that, depending on the interactions of the robot with the object and on the sensors available, explicit communication can be removed from the pushing protocols in cooperative manipulation. This work is complemented by [Rus et al., 1995] that shows several experiments that use different protocols to cooperatively change the pose of objects.

In fact, several different approaches that rely only on sensors to coordinate the robots have been proposed for cooperative manipulation. This is the case of [Kosuge and Oosumi, 1996] that uses a decentralized control algorithm in which the motion command is given to one of the robots (the leader) and the other robots control their trajectory estimating the object motion through their sensors. This same approach has been extended using a group of holonomic robots equipped with body force sensors [Kosuge et al., 1998] and, more recently, using impedance control instead of force/torque sensors [Kume et al., 2001]. Other work that rely only on sensors for cooperative manipulation is [Khatib et al., 1995], in which mobile manipulators (Puma 560 manipulators mounted on Nomadic's XR4000 holonomic bases) equipped with force sensors use a decentralized control structure to manipulate objects. Also, in [Brown and Jennings, 1995], two robots are used to push/steer an object and control is based on the interactions among them and the object. A contribution of that work is the introduction of the term "strong cooperation". It is a synonym for tightly coupled cooperation, meaning that the robots must act in concert to achieve the goal, *i.e.* the strategy for the task is not trivially serializable. Another interesting approach can be found in [Kube and Zhang, 1997], where a group of 31 homogeneous robots are used in a box pushing task. In that work, tasks are decomposed using finite state machines and transitions between discrete states are specified as locally sensed perceptual cues that are highly dependent on the selected sensors and the environment for which the cue was designed [Kube and Zhang, 1996].

It is important to mention that in some of these works, for example [Kosuge and Oosumi, 1996], [Khatib et al., 1995] and others not mentioned here, the main focus is on the manipulation dynamics and not on the coordination mechanisms. Also, a significant number of these approaches are specific to cooperative manipulation and cannot be used for other types of cooperative tasks. But some of the general coordination systems described in the previous section were also used for manipulation tasks. For example, one of the tasks performed with the Alliance architecture is a box pushing task, in which fault tolerance and adaptive control were demonstrated [Parker, 1994]. Similar experiments were presented in [Mataric et al., 1995], where two six-legged robots using explicit communication and a task sharing paradigm were used within the behavior-based framework of [Mataric, 1994]. The Murdoch architecture, mentioned in the previous section, was also used in a box pushing task: in [Gerkey and Mataric, 2002] there is a description of a distributed control system that enables a team of three heterogeneous robots to cooperatively relocate a box to a specified goal, using the task allocation mechanism presented in [Gerkey and Mataric, 2001].

The majority of works in cooperative manipulation (and in cooperative robotics in general) uses distributed approaches to control cooperation. Each robot has its own controller that uses local and global information acquired through sensing and communication to control itself. The distributed approach is more suitable for cooperative robotics because it allows a higher level of independence among the robots. This implies improved scalability and fault tolerance when compared to centralized approaches. One work that applies this approach is presented in [Stilwell and Bay, 1993], where a swarm of ant-like robots is used in a material transportation system. This swarm is composed of several small robots, with no central controller and minimal inter-robot communication. Simulations show the use of this system in the transportation of a palletized load. But combinations of centralized and distributed approaches are also used is some architectures, mainly when global plans are necessary for the execution of the task. This is the case of [Ota et al., 1995], where motion planning is used in an object transportation through an environment with obstacles. A mix of distributed and centralized control is also used in the BeRoSH sys-

tem [Wang et al., 1996], where a central host acts as a leader generating goals and offering some global positioning for the robots.

More recently, different approaches have been proposed for cooperative manipulation. For example, the manipulation using ropes [Donald et al., 2000], that allows a group of robots to reposition and transport several objects together. Other interesting work is the cooperation among a human and a team of robots [Hirata and Kosuge, 2000], where a human applies the motion intention on the object and the robots act like passive casters helping the user. Finally, the cooperative manipulation of long payloads using Mars rovers [Ollennu et al., 2002], that may have a great importance in planetary exploration.

The work developed in the GRASP Laboratory [Sugar and Kumar, 1998a], [Sugar, 1999], and [Sugar and Kumar, 1999], in which a decentralized control system was developed for the coordinated control of mobile manipulators, has served as a base for some of our experiments. In that work, the use of passive compliance (implemented by an actively controlled parallel manipulator) was shown to result in a robust grasp that allows the cooperative transportation of a box by multiple robots. In that framework, the leader plans a trajectory and broadcasts it to the followers, which control their trajectory based on this plan and on feedback from their position, velocity, and sensors.

An extension of the cooperative manipulation task is what we call cooperative search and transportation (also called cooperative search and rescue [Jennings et al., 1997] or object sorting [Lin and Hsu, 1995]). In this task, before manipulating an object in cooperation, the robots must cooperatively locate the object. So, this task is a combination of a loosely coupled task, where the robots search the area independently looking for objects, and a tightly coupled task, in which the robots must manipulate objects in cooperation. This characteristic makes the cooperative search and transportation a very interesting testbed for coordination mechanisms.

2.3 Formal Modeling of Robotic Tasks

In spite of the large number of works dealing with coordination mechanisms and approaches to solve cooperative tasks, very few have considered aspects of modeling and formal verification of cooperative robotics. As mentioned, several coordination mechanisms rely on the behavior-based approach, which imposes difficulties in modeling due to its lack of formalism. As will be explained later in this section, some researchers have applied formal models to the behavior-based control of single robots, but for cooperative approaches not much has been done. One framework that tries to better formalize the behavior-based paradigm in cooperative robotics is described in [Kube and Zhang, 1997], where finite state machines are used to encapsulate behaviors and discrete transitions are controlled by binary sensing predicates. Finite state machines do not model the continuous aspects of the system but can help in the modeling of the discrete part of the cooperation.

An example of a cooperative robotic architecture that uses a more formal approach in its description is the MARS Project at the GRASP Laboratory [Alur et al., 2001]. That architecture uses the Charon framework [Alur et al., 2000a] for the description of multiple agents and their behavioral structure. As will be explained in the next chapter, Charon is a high level modeling language for the specification and simulation of hybrid systems, and can be used for modeling cooperative robotic systems.

The formal modeling of single robot controllers and tasks has received more attention in recent years, even though the number of works in this area is also not very large. Hybrid systems is one of the approaches that has been used by some researchers in order to model robotic tasks for single robots. In [Egerstedt and Hu, 2002] for example, there is a description of how a behavior-based control system for an autonomous robot can be modeled using a hybrid automaton. Other works, such as [Milutinovic and Lima, 2002] and [Wang and Saridis, 1993] have used Petri Nets for modeling robotic tasks.

A very interesting discussion about the specification and formal verification of robotic tasks is presented in [Espiau et al., 1995]. It gives a brief overview about formal modeling and verification of robotic tasks and discusses some approaches that have been used in

this field, including hybrid systems. Also, there is a description of the ORCCAD design environment and two features developed within it for the specification of robotic systems called "Robot Task" and "Robot Procedure". Verification can be performed using these specification features together with tools for logical and temporal verification of certain classes of hybrid systems (timed automata, linear automata, etc). More important, the paper concludes stating that "we believe that all the area of hybrid systems (modeling, programming, formal verification) is a key research domain for the future, the results of which will find particularly relevant applications in robotics". Our hybrid systems modeling of cooperative robotics presented in this thesis is certainly a contribution in this direction.

Chapter 3

Coordination of Multiple Robots

The coordination of multiple robots in the execution of cooperative tasks is one of the fundamental aspects of cooperative architectures. In general, to execute cooperative tasks the robots should synchronize their actions and exchange information. Also, the coordination mechanism should be flexible, allowing the robots to adapt to unexpected events during task execution. Coordination is even more important in the execution of tightly coupled tasks, where each robot depends on the actions of its teammates and the task cannot be completed by a single robot working independently.

In this chapter, we propose a coordination mechanism based on dynamic role assignment. Each robot performs a role that determines its actions during the cooperative task. According to its internal state and information about the other robots and the task received through explicit communication, a robot can dynamically change its role, adapting itself to changes and unexpected events in the environment. We also discuss some important aspects of distributed systems applied to cooperative robotics, such as communication and hierarchy.

3.1 Distributed Systems

Using the computer science terminology, multiple robots working cooperatively can be considered a special type of a distributed system. A distributed system is an asynchronous concurrent system composed by multiple computing systems connected by a communication network [Garg, 1996]. Asynchronous in this case means that computers execute concurrently but do not share the same clock. Distributed systems use mainly message passing to exchange information among distributed processes.

Communication and synchronization are two fundamental aspects in coordination of distributed systems. In fact, a general definition is: coordination = communication + synchronization. Basically, synchronization is necessary when two processes that are executing concurrently need to perform a joint action, such as exchanging data, or when one process should execute an action that depends on the others' actions (for example, some input of a process A depends on the output of a process B). Exclusive accesses to shared resources should also be synchronized (mutual exclusion). One of the ways of synchronizing multiple processes is using communication. Communication is also necessary in coordination mechanisms to exchange required information among the processes. There are several ways of communicating distributed processes. The most common and intuitive is the message passing or, more formally, message oriented communication. This is defined as a form of communication in which the user is explicitly aware of the message and the mechanisms used to send and receive it. A message can be viewed as a collection of typed data objects which is constructed by a process and which is delivered to its final destination. Without going into further details, some characteristics of message passing mechanisms are their use of buffers (buffered vs. unbuffered), the necessity of the sender to wait for a message to be sent (blocking vs. non-blocking), etc. More detailed information can be found in [Mullender, 1993].

Another important concept in distributed systems is the global state of the system. The global state of a distributed system is the union of the states of its individual processes [Babaoglu and Marzullo, 1993]. Given that these processes do not share memory, a process

that wishes to construct a global state must infer the remote components of that state through message exchanges. This can be a challenging task, since the processes do not share the same clock and different processes can construct different global states depending on the sequence of message arrivals. Also, message delays and failures can compromise the correct construction of global states. Therefore, a global state can become obsolete, incomplete, or even inconsistent. Informally, a state is inconsistent if it could never have been constructed by an idealized observer that is external to the system. In general, global states are used to infer properties about the system. Two important properties that can be inferred form the global state are safety and liveness. Informally, safety implies that "nothing bad ever happens", *i.e.*, the system will always remain in a "good" state, while liveness implies that "something good eventually happens", *i.e.*, the computation progresses and the system reaches an objective state (for example terminate correctly) in finite time.

Related to the global state and the verification of properties we have the notion of superimposition. Superimposition is concerned with the use of some high level entity to monitor the execution of a distributed system. Normally this entity is another process in the system that can be responsible for insuring certain properties, taking snapshots, detecting deadlocks, etc. Superimposition allows the separation of concerns in the system because another process will be responsible for these types of monitoring tasks. The basic idea is to try to add functionality to the system without disturbing the desirable properties of the original system [Katz and Gil, 1999].

3.2 Distributed Robots

As mentioned, cooperative robotics can be considered a special case of distributed systems. Instead of several distributed processes, we have several robots that must be synchronized and coordinated. In the next sections we will apply some of the distributed systems aspects presented in the previous section to multi-robot coordination mechanisms and show how we cope with then in our coordination mechanism.

3.2.1 Coordination, Communication, and Synchronization

To execute cooperative tasks, a team of robots must be coordinated. We consider that each robot has a specification of the possible actions that should be performed during each phase of the cooperation in order to complete the task. The coordination mechanism will be responsible for allocating the correct actions to each robot and synchronizing the cooperative execution. The actions performed by each team member must be specified and synchronized considering several aspects, such as robot properties, task requirements, and characteristics of the environment. An important characteristic of distributed robots when compared to general distributed systems in computer science is that robots are physical entities that have a body and act in the real world, having specific dynamics and interacting with other physical entities such as obstacles and other robots. These characteristics are generally known as situatedness and embodiment [Brooks, 1991] and make real robot systems more vulnerable to failures when compared to computer programs. Thus, in addition to organizing the robots and performing synchronization and communication, the coordination mechanisms should provide flexibility and adaptability, coping with failures, dynamic environments, and unpredictable situations.

Inter-robot communication is necessary for any coordination mechanism. Basically, it can be categorized as explicit or implicit. In explicit communication, robots perform some specific act in order to exchange information with the others. The most common method for explicit communication is message passing, in which robots explicitly send information using some kind of wireless communication (wireless ethernet, radio, etc). On the other hand, implicit communication occurs as a consequence of robots actions, exploring the situatedness and embodiment characteristics. Robots use their sensors to observe the behavior and actions of their teammates and the way they change the environment, extracting relevant information from this observation. For example, if one robot observes other carrying an item in a cooperative foraging task, it can infer that it is moving towards the goal, thus having information about the goal direction and the state of its teammate. Both explicit and implicit communication have advantages and disadvantages. Explicit

communication is more powerful and allows the exchange of more detailed information, since various different types of data can be codified as messages. Also, if the communication system is reliable, explicit communication can be used to reduce the uncertainty inherent to the sensors used in implicit communication. Implicit communication is more restricted regarding the amount and type of information exchanged but has advantages such as simplicity, scalability, low power consumption, and robustness to faulty communication environments.

Our coordination mechanism, presented in the next sections, relies on explicit communication to synchronize and exchange information among team members. We implemented two types of explicit communication: synchronous and asynchronous. In synchronous communication, the messages are sent and received continuously in a constant rate, while in asynchronous communication an interruption is generated when a message is received. Synchronous messages are important in situations where the robots must receive constant updates about the state of the others. On the other hand, asynchronous communication is used for coordination when, for example, one robot needs to inform the others about unexpected events or discrete state changes such as the presence of obstacles, robot failures, etc.

As mentioned in the previous section, another fundamental aspect of coordination is synchronization. In cooperative robotics, synchronization is necessary mainly in the execution of tightly coupled tasks, in which robots must tightly coordinate their actions to complete the task. When executing loosely coupled tasks, the robots act more independently from each other and synchronization is not absolutely necessary. In our coordination mechanism, we use explicit communication to synchronize the robots. Basically, when some kind of synchronization is necessary (for example, when two robots should start the cooperative transportation of an object), the robots exchange messages to coordinate their actions.

Our coordination framework is completely decentralized. Each robot has its own controllers and takes its own decisions based on local and global information. In general, each team member has to communicate with other robots to gather information but they nor-

mally do not need to construct a complete global state of the system for the cooperative execution. As mentioned before, the construction of a global state may be necessary to study some general properties of the system. As discussed in the previous section, this can be done by a superimposing entity (also called supervisory system, using a terminology from control theory). In robotics and automation, it is very common to have a human operator supervising the execution or some monitoring processes that can observe the system using sensors and take actions in case of failures. In the present work, our framework does not verify these types of properties, letting this task to some external entity. Using the hybrid systems modeling discussed in the next chapter, it may be possible to perform, to some extent, verification of properties a priori, analyzing the model of the system.

3.2.2 Hierarchy

Another important point in coordination mechanisms is how to hierarchically organize the robots. This organization may be used to define several aspects of the cooperation such as information flow, responsibilities, priorities, etc. The simplest form of organization is without any hierarchy. In this case, all robots are in the same level of coordination, *i.e.*, they do not depend on continuous information from any other teammate and can communicate with any other robot in the group. This organization is very suitable for loosely coupled tasks or other tasks where the robots do not need to exchange much information.

A second possibility is to organize the robots in a leader-follower hierarchy. In this organization the leader normally has a plan or a general knowledge about the task and sends information about its state to the followers, which act in order to cooperate with the leader. The followers can also send messages to the leader, for example, requesting synchronization or treating other coordination issues. A diagram of such organization can be seen in Figure 3.1 that shows one leader and several followers. The leader has a plan and broadcasts information about its state to the followers. The followers have controllers that compute their trajectories based on their own states and information sent by the leader. All robots have coordination modules that exchange messages in order to synchronize

robot actions. Explicit communication composed by both synchronous and asynchronous messages are used in this leader-follower architecture. Synchronous messages, also called data messages, are continuously broadcast by the leader and contain an estimate of its current state. This information can be used by the followers to control their positions or velocities with respect to the leader. Asynchronous messages are exchanged among the robots to coordinate and synchronize the cooperative execution and are called control messages. The experiments presented in Chapter 6 use this kind of hierarchy together with the role assignment mechanism in a cooperative manipulation task.

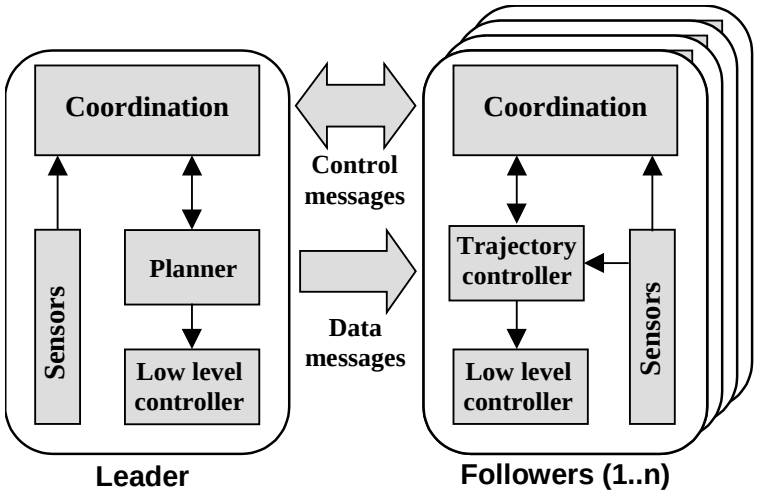


Figure 3.1: Leader-follower architecture.

It is possible to generalize this leader-follower hierarchy to allow the use of multiple leaders. While the team has one designated lead robot, there may be many leaders. A follower can be a leader for another follower as shown in Figure 3.2. This can be thought of as a military hierarchy in which there are several levels of command. These leader-follower interactions or controllers are best described by a directed, acyclic graph (tree) as discussed in [Desai et al., 1999]. Another generalization is to use multiple independent leaders creating a hierarchy that can be represented by a set of different trees. Using the same military analogy, this would be an army composed by the armies of different countries, each one with its leader and independent hierarchy.

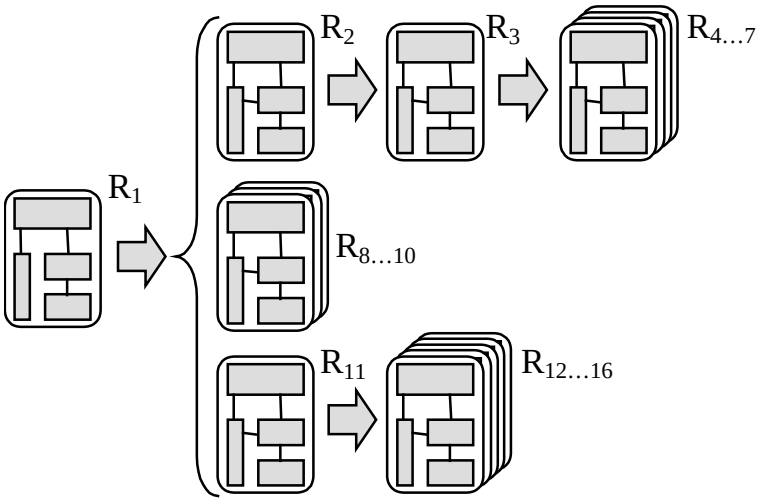


Figure 3.2: Example of an architecture with one team leader (R_1) and three followers that are also leaders (R_2, R_3, R_{11}).

From the communication point of view, the coordination hierarchy can be thought of as a network topology connecting multiple computers. Consequently, the robots' hierarchical organization may define how information will flow inside the team. For example, considering the robots organized in a tree topology, one robot will only be able to communicate with his "father" and his "sons" in the hierarchy, and not with his "siblings" or other robots. This restriction in the communication paths can be very important to reduce the number of messages in teams composed by a large number of robots. Considering a team with n members, in which all robots can communicate with each other, $O(n^2)$ messages may be in the system if all robots broadcast at the same time. Other problems, including physical interference, may also happen in the case of implicit communication if no hierarchy is used. As will be discussed in Chapter 7, communication is one of the problems in the scalability of our coordination system, and this hierarchical organization can help reducing this problem.

Another important point is that the hierarchy should be flexible. During the execution of the task, it may be necessary to change the leader or modify parts of the hierarchy due to unexpected events, such as the presence of obstacles, failures, etc. Also, robots acting

independently from each other in a cooperative task may build temporary hierarchies to perform some specific actions. For example, supposing that the robots have a limited communication range, they can build ad-hoc networks to transmit data to a distant base, as shown in [Pimentel, 2002]. Another example is the cooperative search and transportation presented in Chapter 5, in which robots act independently searching for objects to be retrieved, but must build hierarchies in order to transport them cooperatively. The role assignment mechanism presented in next section allows this dynamic reconfiguration of the hierarchy, allowing robots to adapt to these situations.

3.3 Dynamic Role Assignment

As discussed in the previous sections, robots executing cooperative tasks must be coordinated. We developed a mechanism for the coordination of multiple robots executing cooperative tasks based on dynamic role assignment. Basically, each robot performs a role that determines its actions during the cooperative task. According to its internal state and information about the other robots and the task received through explicit communication, the robot can dynamically change its role, adapting itself to changes and unexpected events in the environment. The implementation of this role assignment mechanism is one of the contributions of this thesis.

Before describing in details the role assignment mechanism, it is necessary to define what is a role in a cooperative task. Webster's Dictionary[1] defines role as: (a) *a function or part performed especially in a particular operation or process* and (b) *a socially expected behavior pattern usually determined by an individual's status in a particular society.* We consider that a role is a function that one or more robots perform during the execution of a cooperative task. Each robot will be performing a role while certain internal and external conditions are satisfied, and will assume another role otherwise. The role will define the behavior of the robot in that moment, including the set of controllers used by the robot,

[1] http://www.webster.com.

the information it sends and receives, and how it will react in the presence of dynamical and unexpected events.

The role assignment mechanism allows the robots to change their roles dynamically during the execution of the task, adapting their actions according to the information they have about the system and the task. Basically, there are three ways of changing roles during the execution of a cooperative task: the simplest way is the **Allocation**, in which a robot assumes a new role after finishing the execution of another role. In the **Reallocation** mechanism, a robot interrupts the performance of one role and starts or continues the performance of another role. Finally, robots can **Exchange** their roles. In this case, two or more robots synchronize themselves and exchange their roles, each one assuming the role of one of the others. Figure 3.3 shows a diagram with the three types of role assignment. The vertical bars inside the robots indicate the percentage of the role execution that has been completed.

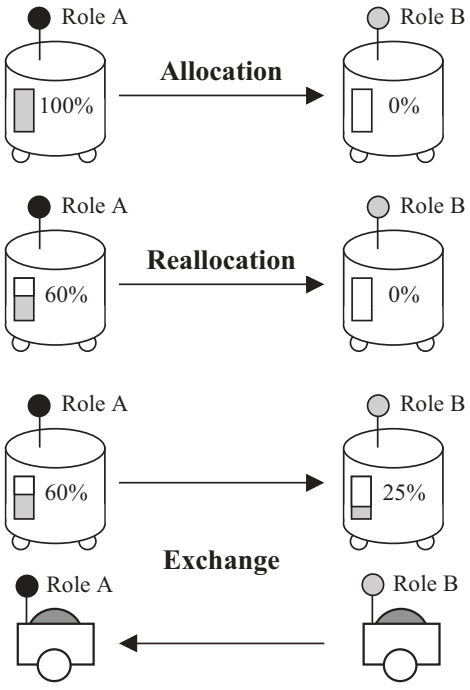


Figure 3.3: Types of role assignment.

The role assignment mechanism depends directly on the information the robots have about the task, the environment and about their teammates. Part of this information, mainly the information concerning the task, is obtained a priori, before the start of the execution. For example, each robot knows a priori what roles it can assume during the execution of the task and when it should change its current role. The rest of the information used by the robots is obtained dynamically during the task execution and is composed by local and global parts. Local information is the robot's internal state and its perception about the environment. Global information contains data about the other robots and their view of the system and is normally received through explicit communication. A key issue is to determine the amount of global and local information necessary for the role assignment. As mentioned, this depends on the type of the task being performed. Tightly coupled tasks require a higher level of coordination and consequently a greater amount of information exchange. On the other hand, robots executing loosely coupled tasks normally do not need much global information because they can act more independently from each other.

An important point is to define when a robot should change its role. In the role allocation, the robot detects that it has finished its role and assumes another available role. The possible role transitions are defined a priori and can be modeled using hybrid systems as will be explained in Chapter 4. In the reallocation process, the robots should know when to give up the current role and assume other. A possible way to do that is to use a function that measures the utility of performing a given role. A robot performing a role r has a utility given by μ_r. When a new role r' is available, the robot computes the utility of executing the new role $\mu_{r'}$. If the difference between the utilities is greater than a threshold τ ($\mu_{r'} - \mu_r > \tau$) the robot changes its role. Function μ can be computed based on local and global information and may be different for distinct robots, tasks and roles. Also, the value τ must be chosen such that the possible overhead of changing roles will be compensated by a substantial gain on the utility and consequently a better overall performance. Chapter 5 shows an example of an utility function used for role reallocation in a cooperative task. The other type of role assignment is the role exchange. In this case, one robot assumes the role of the other. For this, the robots must agree in exchanging

roles and should synchronize the process, which is done using communication. Chapter 6 shows an experiment where two robots exchange roles in order to complete a cooperative task.

Thus, dynamically allocating, reallocating, and exchanging roles in a synchronized manner, the robots are able to work cooperatively in a coordinated fashion, performing the task more efficiently, adapting to unexpected events in the environment, and improving their individual performance in benefit of the team.

Chapter 4

Hybrid Systems Modeling

In this chapter, we describe the use of hybrid systems theory for modeling multi-robot systems in the execution of cooperative tasks. A hybrid system is a dynamical system with both continuous and discrete components. Basically, hybrid systems can be represented by finite state machines augmented with variables that evolve according to continuous differential equations and discrete assignments. We start the chapter giving a brief overview of hybrid systems and hybrid automata. Then we show our approach to model cooperative robotics using hybrid systems and the composition of multiple automata. Finally, we present an example of hybrid systems modeling of a cooperative task.

4.1 Overview of Hybrid Systems

A hybrid system is a dynamical system composed by discrete and continuous states. The execution of a hybrid system can be defined by a sequence of steps: in each step, the system state evolves continuously according to a dynamical law until a discrete transition occurs. These transitions are instantaneous state changes that separate continuous state evolutions [Alur et al., 1995]. Therefore, the state of a hybrid system may change in two ways: due to elapse of time (continuous evolution) or by a switch of the discrete state (discrete transition).

Hybrid systems theory is a very broad research area that has been developed with contributions from both computer science and control theory. Hybrid systems have been used as mathematical models for important and diverse applications such as air traffic management, robotics, biological and medical applications, embedded systems, real-time communication networks, etc [Alur et al., 2000b]. In general, the majority of the systems that are composed by a digital program controlling an analog plant can be modeled using hybrid systems.

Several aspects of hybrid systems have been studied in recent years and an interesting overview of the field can be found in a special issue of the Proceedings of the IEEE [Antsaklis, 2000]). One of the main areas of research is modeling, *i.e.*, how to model and describe real systems composed by discrete and continuous dynamics using hybrid systems theory. Other important research area is the analysis of hybrid systems, mainly stability and reachability analysis. Common problems regarding stability include finding conditions that guarantee the stability of the system for any sequence of discrete transitions, or designing a switching sequence that can stabilize a set of unstable equations. On the other hand, reachability is concerned with verifying if some state is reachable from an initial set of conditions. This is important, for example, to obtain formal guarantees about correction of the system or to detect if the execution leads to some undesirable state.

4.2 Hybrid Automata

4.2.1 Description

Normally a hybrid system can be modeled as a finite state machine equipped with a set of variables. Discrete states (control locations) contain evolution laws and the values of the variables change according to these laws while the system is in a specific discrete state. The discrete transitions of the system are labeled with a set of guards and assignments. A transition is enabled when the boolean condition of its guard is satisfied. When a transition occurs, the assignment associated with that transition is executed, possibly modifying the

values of the variables. Additionally, each control location is labeled with an invariant condition that must hold whenever the system is executing in that location.

One possible formal description of this finite state machine is a hybrid automaton. A hybrid automaton is a finite automaton augmented with a finite number of variables that can change continuously, as specified by differential equations, or discretely, according to specific assignments. A hybrid automaton H can be defined as:

$$H = \langle Q, V, E, f, Inv, Init \rangle,$$

where:

- $Q = \{q_1, q_2, \ldots, q_n\}$ is the set of discrete states of the system, also called *control modes* or *control locations*.
- V is a finite set containing the variables of the system and can be composed by discrete (V_d) and continuous (V_c) variables: $V = V_d \cup V_c$. Each variable $x \in V$ has a value that is given by a function $\nu(x)$. This is called *valuation* (ν) of the variables. Thus, at any moment, the state of the system is given by a pair (q, ν), composed by the discrete state and the valuation of the variables.
- Discrete transitions between pairs of control modes are specified by control switches represented by E (also called *edges*). Each transition has an associated predicate g that is called a guard or jump condition. The discrete transition can only be taken (transition is enabled) if its predicate g is satisfied. Each transition may also have a reset statement r that changes the value of some variable or perform some action during a discrete transition. Finally, control switches can be tagged with a label $a \in \Sigma$. This label can be also called *event* and is important for synchronization issues in the composition of multiples automata. Thus, each discrete transition $e \in E$ can be represented by a tuple $\langle q_i, q_j, g, r, a \rangle$ where $q_i \in Q$ and $q_j \in Q$ are the source and destination states, g is the guard, r is the reset statement and $a \in \Sigma$ is the label.

- The dynamics of the continuous variables are determined by the flows f, generally described as differential equations inside each control mode.

- Invariants (Inv) are predicates related to the control modes. The system can stay in a certain control mode q while its invariant is satisfied and must left the mode when it becomes invalid. If the invariant becomes false and there are no transitions enabled for the current control mode, the system is considered to be in a deadlock.

- $Init$ is the set of initial states of the system. Each initial state is composed by a pair $(q, \nu(X))$, where $q \in Q$, and $X \subseteq V$.

Different representations of hybrid automata may exist depending on the author and the context in which the definition is applied. For example, some authors consider that the set V can only be composed by continuous variables and call it X, representing the continuous state of the system. Others represent the guards, reset statements and labels, as specific components G, R and Σ decomposed from the set E. Some other descriptions of hybrid automata can be found in [Henzinger, 1996] and [Alur et al., 1995] for example. In spite of these small differences, the general idea of the hybrid automaton does not change.

Also, there are some restricted classes of automata that can be used to model some specific systems. One example the timed automata [Alur and Dill, 1994] that is used to represent discrete systems augmented with clocks. The clocks are continuous variables representing the time within each discrete state. A timed automaton can be viewed as a hybrid automaton where all continuous equations have the form $\dot{x} = 1$. A simple example of a system that can be modeled using a timed automata is a traffic light that switches between red and green every 60 seconds. Another important class is linear automata [Henzinger et al., 1997], that constrains flows, invariants, and transitions to linear equations. These special classes of automata are important because they allow the development of mathematical proofs and automatic verification tools that help in the study of properties such as stability, reachability, etc. This is much harder or even impossible in a hybrid automaton due to its generality and complexity.

4.2.2 Example

A classical example of a hybrid automaton is the model of a thermostat. As shown in Figure 4.1 [Henzinger, 1996], a hybrid automaton can be pictorially represented by a directed graph, in which discrete modes are represented by vertices and discrete transitions by edges. In Figure 4.1, the system starts in the Off mode with the temperature x set to 20 degrees. In this mode, the temperature falls according to the differential equation $\dot{x} = -0.1x$. Due to the jump condition $x < 19$, the heater may go to the On state as soon as the temperature falls below 19 degrees. Additionally, according to the invariant condition $x \geq 18$, at the latest the heater will move to the On state when the temperature falls below 18 degrees. In the On control mode, the temperature increases according to the equation $\dot{x} = 5 - 0.1x$ and the control jumps to the Off mode when the temperature is between 21 and 22 degrees. It is important to note that the hybrid automaton allows nondeterminism in the execution of the system. A discrete transition can be taken in any moment provided that its guard is valid. To force determinism in the execution, guards and invariants must be synchronized in such a way that a transition will become enabled in the exact moment that an invariant is violated. In this example, this would be possible by setting the guards to $x \leq 18$ and $x \geq 22$.

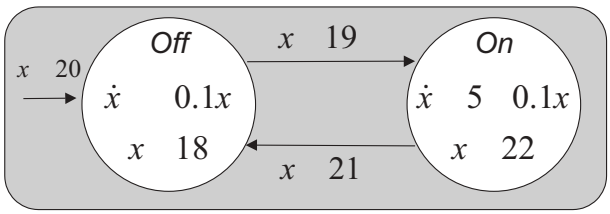


Figure 4.1: Hybrid automaton modeling a thermostat.

4.3 Architectural and Behavioral Hierarchies

Recently, some important concepts from software engineering have been applied to the modeling of hybrid systems. The basic idea is to try to use concepts like modularization, encapsulation, hierarchy, etc, to build models that are easier to understand, implement, extend, and reuse in other systems. An interesting effort in this direction is the Charon language [Alur et al., 2000a], that is being developed at the University of Pennsylvania. Charon is a modeling language for the specification and simulation of hybrid systems. A hybrid system in Charon is described by a set of parallel agents that communicate through a set of shared variables in an asynchronous way. The behavior of each agent in Charon is given by a set of control modes that are connected by a set of transitions. Each mode has a certain continuous behavior given by a set of algebraic and differential equations. Invariants and guards control the activation of modes and transitions. In a sense, one can consider that each agent's behavior is modeled by a hybrid automaton. An important point is that these modes may be grouped in a hierarchical way, with higher level modes being composed by submodes. This is called *behavioral hierarchy* of the system. Agents may also be grouped hierarchically creating more sophisticated agents from more primitive (atomic) ones. Variables representing the state of an agent may be hidden from others, encapsulating the data. The grouping of agents into compound agents gives the *architectural hierarchy* of the hybrid system. For example, as mentioned in [Alur et al., 2000a], an agent representing a robot can be modeled as a composition of a sensing agent, a controller agent, etc.

Some of these concepts are used in our hybrid systems modeling of cooperative robotics, described in the next section. Firstly, the roles can be organized creating a behavioral hierarchy. A role can be composed by several subroles, each one with its own behavior. This is the case, for example, of some of the experiments presented in Chapter 6, in which a high level role called Transport is divided in the subroles Lead and Follow. The behavioral hierarchy helps organizing the roles, encapsulating the variables and continuous equations inside each control mode and creating high level roles from the composition of lower level ones. Agents can also be organized in hierarchies. But our hierarchical organization is

different from the one explained in the previous paragraph. We do not have the concept of agents being composed by other agents. All agents in our methodology are high level entities. As explained in Chapter 3, the hierarchy defines, among other things, how the information flows among the agents. We can have an independent parallel composition, a leader-follower hierarchy, etc, and it can be dynamically changed during the execution of the task.

4.4 Modeling a Cooperative Multi-Robot System

In general, a cooperative multi-robot system can be described by its state ($\boldsymbol{X}$), which is a composition of the states of the robots:

$$\boldsymbol{X} = [\boldsymbol{x_1}, \boldsymbol{x_2}, \ldots, \boldsymbol{x_n}]^T, \quad \dot{\boldsymbol{X}} = F(\boldsymbol{X}).$$

Considering a simple control system, the state of each robot varies as a function of its continuous state ($\boldsymbol{x}$) and the input vector ($\boldsymbol{u}$). Also, each robot may receive information about the rest of the system ($\hat{\boldsymbol{z}}$) that can be used in the controller. This information is basically composed by estimates of the state of the other robots that are received mainly through communication. We use the hat ($\hat{}$) notation to emphasize that this information is an estimate because the communication can suffer delays, failures, etc. Using the role assignment mechanism, in each moment each robot will be controlled by a different continuous equation according to its current role in the task. Therefore, we use the subscript q to indicate the current role of the robot. Following this description, the state equation of each robot during the execution of the task can be defined as:

$$\dot{\boldsymbol{x}} = f_q(\boldsymbol{x}, \boldsymbol{u}, \hat{\boldsymbol{z}}).$$

The equations shown above model the continuous behavior of each robot and consequently the continuous behavior of the team during the execution of a cooperative task.

These equations, together with the roles, role assignments, variables, communication and synchronization can be better understood and formally modeled using hybrid systems. As will be explained in the remainder of this chapter, we use a hybrid automaton to model each robot and the composition of these automata to model the cooperative task execution. This is one of the main contributions of this thesis.

Our basic idea to model cooperative robotics using hybrid systems is to represent each role as a control mode[1] of the hybrid automaton. Therefore, internal states and sensory information within each role can be specified by continuous and discrete variables of the automaton. The variables are updated according to the equations inside each control mode (flows) and reset statements of each discrete transition. The role assignment is represented by discrete transitions and the invariants and guards define when each robot will assume a new role. Finally, we model the cooperative task execution using a parallel composition of several automata, as will be explained in the next section. Table 4.1 summarizes the mapping from cooperative robotics to hybrid systems.

Cooperative robotics	**Hybrid systems**
Roles	Control modes
Discrete and continuous information	Variables
Controllers	Flows
Role assignment	Discrete transitions, guards, and invariants
Communication	Send and receive actions, self transitions
Cooperative execution	Parallel composition of hybrid automata

Table 4.1: Modeling cooperative robotics using hybrid systems

Communication among robots can also be modeled in our hybrid systems framework. As explained in Chapter 3, we use a message passing mechanism to exchange information among the agents. To model this message passing in a hybrid automaton, we consider that there are communication channels between agents and use the basic operations send and receive to manipulate messages. The basic syntax is *send(channel, value)* to send a

[1]Since we are using control modes to represent roles, in the remainder of this thesis we use the terms roles, modes and control modes with the same meaning.

message containing a certain value in a channel and *receive(channel, variable)* to receive a message and put its value in a local variable. In fact, the value can be some kind of object or record that groups several values together. Also, messages are implicitly tagged with a message type, to insure that the correct action will be taken for each message that arrives. In the hybrid automaton, messages are sent and received in discrete transitions. These actions are modeled in the same way as assignments of values to variables (reset statements). As will be shown in the example, it is very common to use a self transition, *i.e.*, a transition that does not change the discrete state, to receive and send messages. Discrete transitions are instantaneous and we consider that there are no delays or failures in the communication. We also consider that when a message is sent, it is immediately received before any continuous evolution of the automata.

We have chosen hybrid systems in order to represent cooperative robotics for two main reasons. The first one is that hybrid systems provide a simple and logical way for modeling the cooperative execution of tasks by multiple robots. To model cooperative robotics, it is necessary to represent both continuous and discrete dynamics, together with synchronization, communication, etc. All these aspects are covered in hybrid systems modeling. Also, as described in the previous paragraph, the mapping from our role assignment mechanism to the hybrid automaton is relatively simple. Consequently, using hybrid systems we have a simple, structured, and formal way of reasoning about, representing, and implementing multi-robot cooperation. The second main reason is that hybrid systems are backed by a powerful theory, which may allow the development of formal proofs about some aspects of the cooperative execution. For example, through an analysis of the hybrid systems model of a cooperative task it may be possible to detect deadlock states, test reachability of undesirable or goal states, and study stability of the cooperative system. As will be mentioned in the next sections, methodologies for obtaining some of these formal results are already available for some restricted classes of automata. As hybrid systems theory and computer power evolves, we can expect that in the future we will have more advanced and efficient mechanisms for obtaining formal results from models based on general classes of hybrid automata.

In this thesis we focus only on the modeling of cooperative robots using hybrid systems. The use of techniques for formally studying stability, reachability and other properties of cooperative robots modeled as a composition of hybrid systems is certainly an important but very large research area, and is left for future work.

4.5 Composition of Hybrid Systems

As discussed in the previous section, the execution of a cooperative task by multiple robots can be modeled using a parallel composition of several automata, one for each robot. Our approach to formally describe the composition of hybrid automata is based on the composition of transition systems. Basically, we can consider that a transition system is the discrete part of a hybrid automata, *i.e*, the graph (Q, E) of the automata without considering variables, invariants, etc.

More formally [Alur and Dill, 1994], a transition system S is a tuple $\langle Q, Q^0, \Sigma, E\rangle$ where Q is the set of states, $Q^0 \subseteq Q$ is a set of initial states, Σ is a set of labels (or events), and $E \subseteq Q \times \Sigma \times Q$ is a set of transitions. For a transition $\langle q, l, q'\rangle$ in E we write $q \xrightarrow{l} q'$, where q is the source, q' is the destination and l is the label. The system starts in a initial state and if there is a transition $q \xrightarrow{l} q' \in E$, the system can change its state from q to q' on event l. Two or more transition systems can be combined in a new transition system. Let $S_1 = \langle Q_1, Q_1^0, \Sigma_1, E_1\rangle$ and $S_2 = \langle Q_2, Q_2^0, \Sigma_2, E_2\rangle$ be two transition systems. The parallel composition $S_1 \parallel S_2$ is the transition system $\langle Q_1 \times Q_2, Q_1^0 \times Q_2^0, \Sigma_1 \cup \Sigma_2, E'\rangle$. A state of the composition is a pair (q_1, q_2) with $q_1 \in Q_1$ and $q_2 \in Q_2$. The transitions E' from $S_1 \parallel S_2$ are labeled with symbols from $\Sigma_1 \cup \Sigma_2$. A transition of the form $(q_1, q_2) \xrightarrow{l} (q_1', q_2')$ is part of E' iff either (i) $l \in \Sigma_1 \cap \Sigma_2$, $q_1 \xrightarrow{l} q_1' \in E_1$ and $q_2 \xrightarrow{l} q_2' \in E_2$, or (ii) $l \in \Sigma_1 \setminus \Sigma_2$, $q_1 \xrightarrow{l} q_1' \in E_1$ and $q_2' = q_2$ or (iii) $l \in \Sigma_2 \setminus \Sigma_1$, $q_2 \xrightarrow{l} q_2' \in E_2$ and $q_1' = q_1$. The labels that belong to both Σ_1 and Σ_2 are used for synchronization of the parallel system. For example, if the label l belongs to both Σ_1 and Σ_2, the transition $(q_1, q_2) \xrightarrow{l} (q_1', q_2')$ will only be taken if the transitions $q_1 \xrightarrow{l} q_1'$ and $q_2 \xrightarrow{l} q_2'$ of the two original automata are executed simultaneously.

Based on the composition of transitions systems, we propose an algorithm for composing

multiple automata in order to represent the execution of cooperative tasks by multiple robots. Let $H_1 = \langle Q_1, V_1, E_1, f_1, Inv_1, Init_1 \rangle$ and $H_2 = \langle Q_2, V_2, E_2, f_2, Inv_2, Init_2 \rangle$ be two hybrid automata. Some considerations have to be made before defining the composition. Firstly, we assume that the sets of variables V_1 and V_2 are disjoint. This is a reasonable assumption since there are no shared variables. The two automata can share information, but this information will be stored in different variables and gathered in different ways. Another consideration is that, similarly to transition systems, two transitions from E_1 and E_2 labeled with the same label l are taken at the same time in both automata. This is a way of synchronizing the execution of both automata and can be done, for example, using communication. With these considerations, the parallel composition $H_1 \parallel H_2$ of H_1 and H_2 is the hybrid automaton:

$$H_1 \parallel H_2 = \langle Q_1 \times Q_2, V_1 \cup V_2, E', f_1 \cup f_2, Inv', Init' \rangle.$$

The control modes of the compound automaton are pairs (q_1, q_2), where $q_1 \in Q_1$ and $q_2 \in Q_2$, and the set of variables of $H_1 \parallel H_2$ are the union of the sets V_1 and V_2 from both automata. The discrete transitions (E') are defined based on the transitions of both automata, in the same way as for transition systems:

1. if there is a transition labeled l on both automata ($l \in \Sigma_1 \cap \Sigma_2$), for every transition $\langle q_1, q_1', g_1, r_1, l \rangle$ in E_1 and $\langle q_2, q_2', g_2, r_2, l \rangle$ in E_2, E' contains the transition $\langle (q_1, q_2), (q_1', q_2'), g_1 \wedge g_2, r_1 \cup r_2, l \rangle$;

2. for $l \in \Sigma_1 \setminus \Sigma_2$, for each transition $\langle q_1, q_1', g_1, r_1, l \rangle$ in E_1 and every $q \in Q_2$, E' contains the transitions $\langle (q_1, q), (q_1', q), g_1, r_1, l \rangle \, \forall q \in Q_2$;

3. for $l \in \Sigma_2 \setminus \Sigma_1$, for each transition $\langle q_2, q_2', g_2, r_2, l \rangle$ in E_2 and every $q \in Q_1$, E' contains the transition $\langle (q, q_2), (q, q_2'), g_2, r_2, l \rangle \, \forall q \in Q_1$.

The flows f' of $H_1 \parallel H_2$ are simply the union of the flows from H_1 and H_2: $f' = f_1 \cup f_2$. Since the variable sets V_1 and V_2 are disjoint, the continuous update in each compound

control mode (q_1, q_2) is simply the union of the continuous updates from both q_1 and q_2. In the same way, the invariant of each compound mode (q_1, q_2) is the conjunction (logical and) of the invariants of both q_1 and q_2: $Inv'(s_1, s_2) = Inv_1(s_1) \wedge Inv_2(s_2)$. This means that the execution of the automaton $H_1 \parallel H_2$ can stay in control mode (q_1, q_2) while the invariants of both q_1 and q_2 are true. Finally, the initial state set $Init'$ of $H_1 \parallel H_2$ is a composition of $Init_1$ and $Init_2$. In general, a initial state set $Init$ is composed by a set of discrete states and valuations of some of the variables: $Init = \langle Q^0, \nu(X) \rangle$, where $Q^0 \subseteq Q$, $X \subseteq V$. Therefore, the composition of $Init_1 = \langle Q_1^0, \nu(X_1) \rangle$ and $Init_2 = \langle Q_2^0, \nu(X_2) \rangle$ is given by $Init' = \langle Q_1^0 \times Q_2^0, \nu(X_1 \cup X_2) \rangle$.

4.6 Example: Modeling Cooperative Manipulation

To demonstrate the hybrid automata composition described in the previous section we modeled a task in which two robots coordinate themselves to transport a box in cooperation. We consider that the transportation is performed in one dimension and each robot i has an estimate of its position (x_i) and the position of the box (x_{bi}). In fact, they need only to know their distance to the box, which can be obtained using some kind of range sensor. We also consider that the robots can exchange messages using explicit communication. The robots use a very simple kinematic model, in which the only input is the linear velocity $(\dot{x} = u)$. Figure 4.2 shows a diagram of this task.

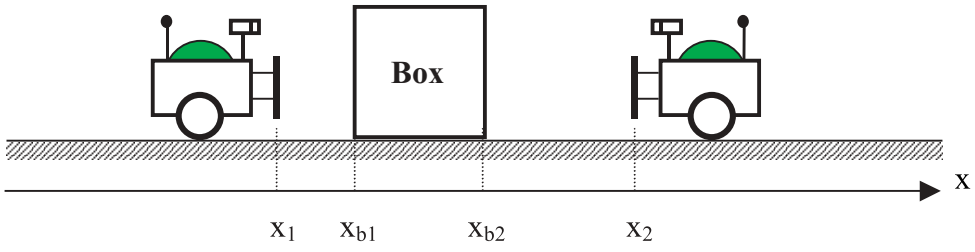


Figure 4.2: Diagram of two robots transporting a box in one dimension.

Before formally describing the behavior of the robots using hybrid automata, let us

give a brief overview of the execution. The robots can be in one of four discrete states or roles: Approach, Wait, Transport, and Lost. Each one starts in the Approach mode, where it uses a proportional controller in order to get closer to the box. When the box is close enough, the robot switches to the Wait mode and sends a message to the other robot informing that its approach phase is complete. Both the Approach and Wait states have self transitions in order to receive this message from the other robot when it is available. When both robots have approached the box, they synchronously start the Transport role, in which they cooperate to move the box. In this mode, one of the robots (robot_1) is the leader and has a plan that sets its velocity. The follower (robot_2) estimates the leader's velocity (v_l) (through sensing or communication, for example) and sets its velocity to the estimated value. If one of the robots loses contact with the box, it sends a message to the other robot and switches to the Lost mode. The other robot will receive this message, send an acknowledgment and both robots will synchronize and switch to the Approach mode, restarting the cycle. Note that we do not use any kind of role exchange in this example, only role allocations and reallocations.

The execution of each robot can be formally modeled using a hybrid automaton. The automaton $H_1 = \langle Q_1, V_1, E_1, f_1, Inv_1, Init_1 \rangle$ for robot_1 is described below and depicted in Figure 4.3. The variables of H_1 include the position of the robot and the box, and three boolean conditions that indicate if the other robot has approached, lost the box, or sent an acknowledgment message. Some special transitions that do not change the discrete state of the automaton (self transitions) are used to set these conditions according to the messages received from the other robot. One example is the transition $\langle A_1, A_1, \text{MsgAvailable}(\text{C}_{21}), \text{Receive}(\text{C}_{21}, \text{DockOk}_2), \phi \rangle$, that sets the variable DockOk_2 when a message from robot 2 is available in channel C_{21}. In the description below we use a short notation for the discrete transitions, showing only the source and destination states $\langle q_i, q_j \rangle$. In figure 4.3, it is possible to see the guards, actions, and labels of each transition: the guards are shown in normal font, the actions in bold and the labels in italic. Some of the transitions may not have all of these components. For example, following the complete notation, the transition $\langle A_1, W_1 \rangle$ is in fact $\langle A_1, W_1, |x_1 - x_{b1}| == 0, \text{Send}(\text{C}_{12}, \text{True}), \phi \rangle$,

where ϕ indicates that there is no label. Flows and invariants are set according to the behavior explained in the previous paragraph, and the execution starts in the Approach mode with the boolean conditions set to false.

- $Q_1 = \{A_1, W_1, L_1, T_1\}$
- $V_1 = \{x_1, x_{b1}, \text{DockOk}_2, \text{LostBox}_2, \text{ACKrecv}_2\}$
- $E_1 = \{\langle A_1, A_1\rangle, \langle A_1, W_1\rangle, \langle W_1, W_1\rangle, \langle W_1, T_1\rangle, \langle T_1, T_1\rangle, \langle T_1, L_1\rangle, \langle T_1, A_1\rangle, \langle L_1, L_1\rangle, \langle L_1, A_1\rangle\}$
- $f_1 = \{\dot{x}_1 = k(x_1 - x_{b1}), \dot{x}_1 = 0, \dot{x}_1 = f(t)\}$
- $Inv_1 = \{|x_1 - x_{b1}| > 0, \text{DockOk}_2 == \text{F}, (|x_1 - x_{b1}| > 0) \wedge (\text{LostBox}_2 == \text{F}), (\text{ACKrecv}_2 == \text{F})\}$
- $Init_1 = \{A_1, \text{DockOk}_2 = \text{F}, \text{LostBox}_2 = \text{F}, \text{ACKrecv}_2 = \text{F}\}$

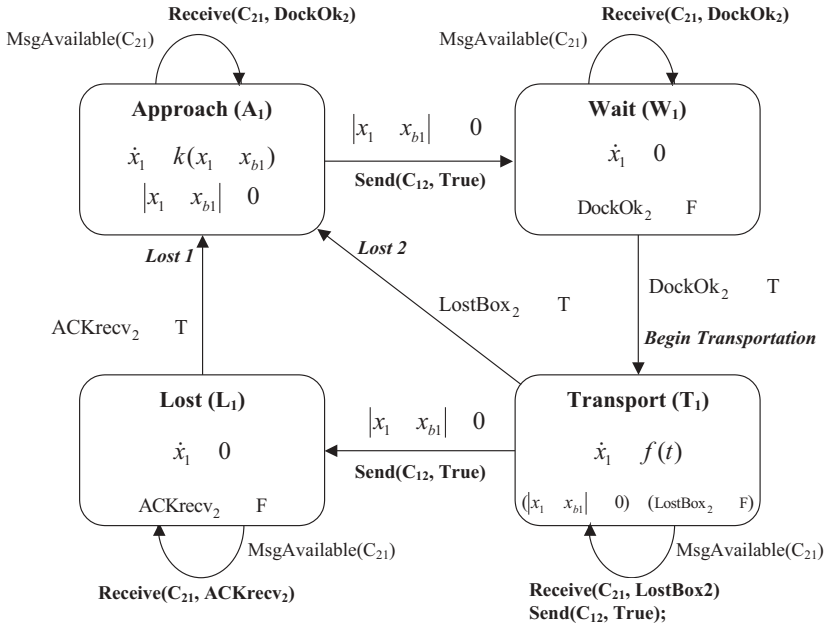


Figure 4.3: Hybrid automaton for robot 1.

The automaton $H_2 = \langle Q_2, V_2, E_2, f_2, Inv_2, Init_2\rangle$ for robot$_2$ is very similar to H_1 and is shown in Figure 4.4. It is basically a copy of H_1 with the variables and communication

channels renamed (from 1 to 2 and 2 to 1). The basic difference is in the dynamic equation of the Transport mode. As mentioned, robot 2 is the follower and sets its velocity to be equal to the leader's velocity: $\dot{x}_2 = \hat{v}_l$. The formal description of the automaton is show below:

- $Q_2 = \{A_2, W_2, L_2, T_2\}$
- $V_2 = \{x_2, x_{b2}, \text{DockOk}_1, \text{LostBox}_1, \text{ACKrecv}_1\}$
- $E_2 = \{\langle A_2, A_2\rangle, \langle A_2, W_2\rangle, \langle W_2, W_2\rangle, \langle W_2, T_2\rangle, \langle T_2, T_2\rangle, \langle T_2, L_2\rangle, \langle T_2, A_2\rangle, \langle L_2, L_2\rangle, \langle L_2, A_2\rangle\}$
- $f_2 = \{\dot{x}_2 = k(x_2 - x_{b2}), \dot{x}_2 = 0, \dot{x}_2 = \hat{v}_l\}$
- $Inv_2 = \{|x_2 - x_{b2}| > 0, \text{DockOk}_1 == \text{F}, (|x_2 - x_{b2}| > 0) \wedge (\text{LostBox}_1 == \text{F}), (\text{ACKrecv}_1 == \text{F})\}$
- $Init_2 = \{A_2, \text{DockOk}_1 = \text{F}, \text{LostBox}_1 = \text{F}, \text{ACKrecv}_1 = \text{F}\}$

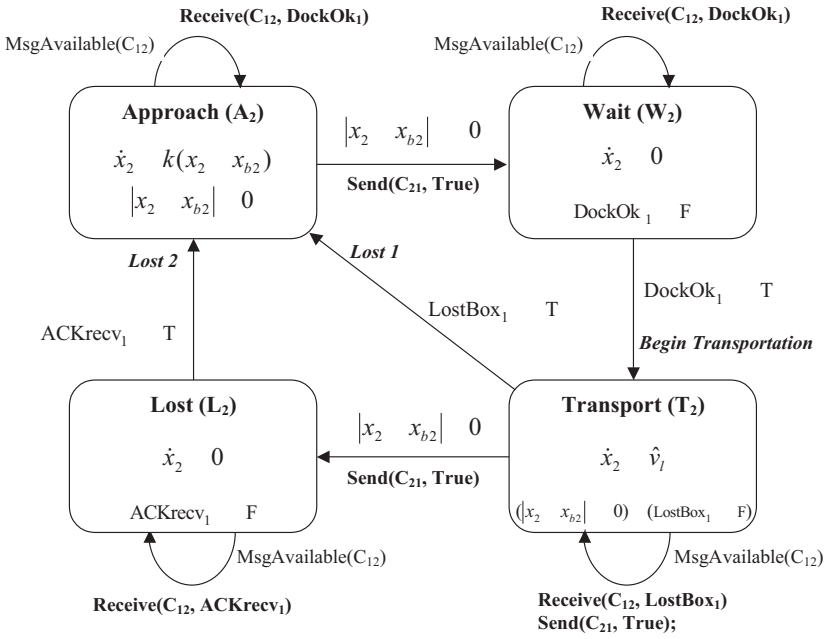


Figure 4.4: Hybrid automaton for robot 2.

The composition of H_1 and H_2 represents the cooperative execution of the task by the two robots. The hybrid automaton $H_1 \parallel H_2$ is built following the rules explained in the previous section. Figure 4.5 shows part of this automaton. In fact, $H_1 \parallel H_2$ is composed by 16 control modes, the product of the modes of the two automata ($Q_1 \times Q_2$), but Figure 4.5 only shows the 8 control modes that are reachable from the initial state of the system. To simplify the figure, we do not show the control modes (and consequently the related transitions, invariants, etc.) that are not reachable from the initial state.

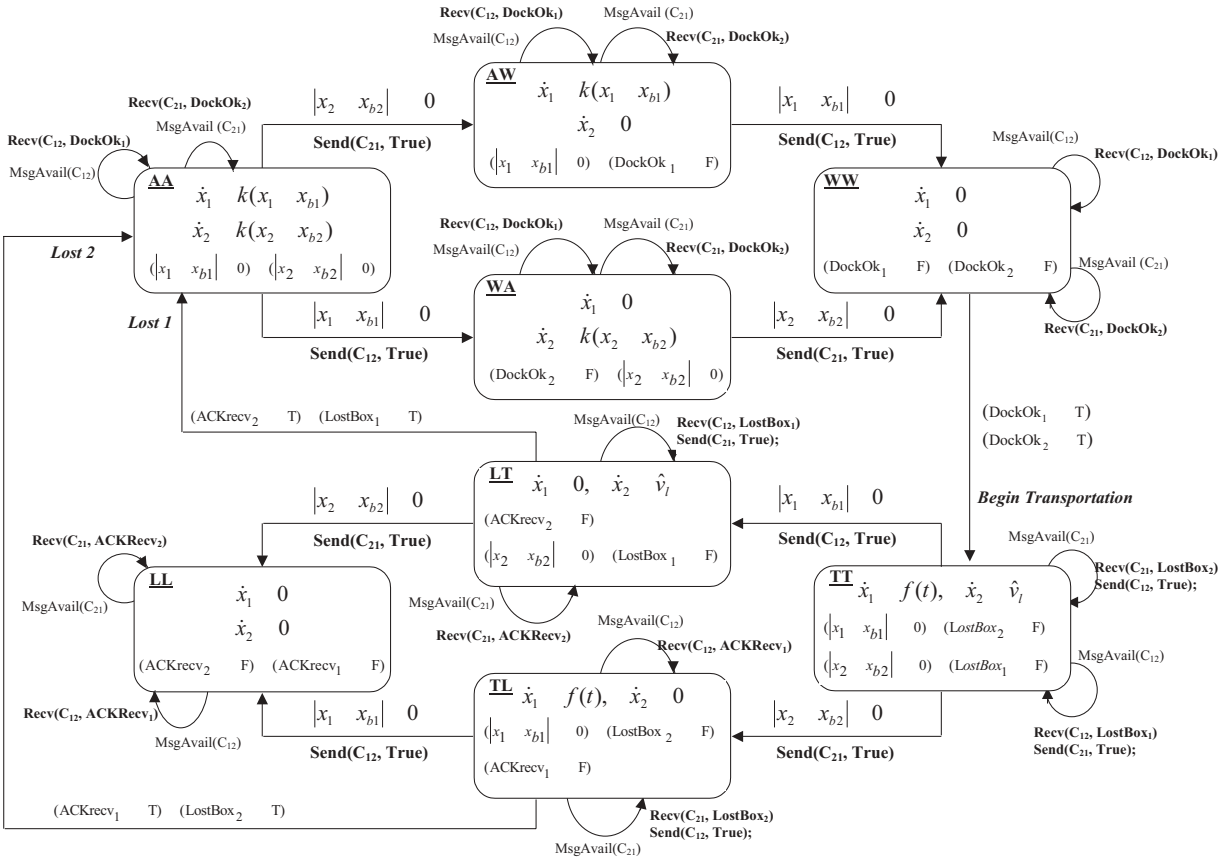


Figure 4.5: Compound automaton.

The composition of control modes, invariants and flows is relatively simple, as explained in the previous section. The major challenge is to define the discrete transitions of the new automaton. According to the composition rules, there are three situations: (i) when there

is a transition labeled l in E_1 but not in E_2, (ii) when there is a transition labeled l in E_2 but not in E_1, and (iii) when there is a transition labeled l in both E_1 and E_2. For situation (i), a transition $q_i \rightarrow q_j$ from mode q_i to mode q_j in E_1 generates a transition $(q_i, q) \rightarrow (q_j, q)$ in the compound automaton for each mode $q \in Q_2$. To exemplify, let us consider the transition $\langle A_1, W_1, |x_1 - x_{b1}| == 0, \text{Send}(\text{C}_{12}, \text{True}), \phi\rangle \in E_1$. In our modeling, we label only the transitions that are part of both automata. Transitions without labels are unique to H_1 or H_2. In the compound automaton, this transition should appear four times connecting the modes: AA $\rightarrow$ WA, AW $\rightarrow$ WW, AT $\rightarrow$ WT, AL $\rightarrow$ WL [2]. Since we are showing only the reachable modes, this transition appear two times in the automaton of Figure 4.5. The same thing happens for situation (ii) when there is a transition in H_2 that is not present in H_1. For example, the transition $\langle T_2, L_2, |x_1 - x_{b2}| > 0, \text{Send}(\text{C}_{21}, \text{True}), \phi\rangle \in E_2$ turns into the transitions AT $\rightarrow$ AL, WT $\rightarrow$ WL, TT $\rightarrow$ TL, LT $\rightarrow$ LL in the automaton $H_1 \parallel H_2$.

The other possible situation (iii) is when there is a transition labeled l in both H_1 and H_2. In this case, as explained in the previous section, a single transition is generated in the compound automaton. For example, the transitions $\langle \text{W}_1, \text{T}_1, \text{DockOk}_2 == \text{T}, \phi, \textit{Begin Transportation}\rangle \in E_1$ and $\langle \text{W}_2, \text{T}_2, \text{DockOk}_1 == \text{T}, \phi, \textit{Begin Transportation}\rangle \in E_2$ results in the transition $\langle \text{WW}, \text{TT}, \text{DockOk}_1 == \text{T} \wedge \text{DockOk}_2 == \text{T}, \phi, \textit{Begin Transportation}\rangle$ in the compound automaton. As mentioned, these labeled transitions are used for synchronization. In the modeling, we consider that transitions that have the same label in both automata are taken at the same time. This is the case, for example, of the *Begin Transportation* transition that synchronize the start of the transportation. Another example is the *Lost 1* transition, in which both robots go to the Approach mode when robot_2 receives the information that robot_1 have lost the box and robot_1 receives the acknowledgment.

In real implementations, synchronization is obtained using explicit communication. When a robot receives a certain message or acknowledgment, it sets some of its variables

[2] To simplify the notation, we use only two letters for the names of the compound modes. The first letter is the mode of Q_1 and the second the mode of Q_2. For example the mode AW in $Q_1 \times Q_2$ is in fact the mode composed by A_1 and W_2.

and switches to another discrete state. In the model presented in this example, we consider that a command to send a message is automatically followed by one to receive the message, before any continuous update of the automaton. If this condition is not followed, some undesirable situations may happen. For example, consider the transition AW $\rightarrow$ WW in the model of Figure 4.5. In mode AW, robot_2 is waiting while robot_1 is approaching the box. Suppose that the variable DockOk_2 of robot_1 is set to true, considering that robot_2 has send a message during transition AA $\rightarrow$ AW and robot_1 has received this message in the self transition of mode AW. When the transition AW $\rightarrow$ WW is executed, a message is sent through channel C_{12} to inform robot_2 that robot_1 has finished its approach phase. If this message is not immediately received in mode WW (*i.e.* the guard $\text{MsgAvail}(\text{C}_{12})$ is not true), the system is deadlocked, since the invariant of mode WW is false and there is no transition enabled for this mode.

Another example of deadlock is the mode LL of the compound automaton. The mode LL is a sink, since there are no transitions leaving from it. In normal situations, the system does not go to this mode from modes LT or TL (the transitions *lost 1* and *lost 2* to the AA mode are taken instead), but if there is some delay between the sending of a message and the receiving of an acknowledgment and both robots lose contact with the box in this interval this state can be reached. In fact, these deadlock situations can be avoided by introducing some extra modes with the objective of waiting for specific messages and synchronizing the automata. In the example presented here, we preferred to avoid this extra states in order to make the automaton less complex.

Chapter 5

Simulations

This chapter presents some simulations that we have performed using the role assignment mechanism and the hybrid systems modeling described in the previous chapters. Firstly, we give an overview of the simulator developed for cooperative robotics and then we show the simulations of two cooperative tasks, namely cooperative manipulation and cooperative search and rescue.

5.1 MuRoS - A Multi-Robot Simulator

Good simulation tools are very important in the study of cooperative robotics. Normally, it is difficult and expensive to obtain and maintain multi-robot teams, specially when a large number of robots (say, 10 or more) is required. Additionally, multi-robot simulators allow researchers to achieve results rapidly and to implement and test different control, modeling, and coordination approaches before starting real implementations. In order to perform simulations of cooperative robots, we have developed MuRoS, a software tool for simulating multi-robot teams in the execution of cooperative tasks. It allows the simulation of several multi-robot applications such as cooperative manipulation, formation control, foraging, etc. Both loosely coupled and tightly coupled tasks can be simulated making MuRoS a very useful tool in the study of cooperative robotics.

Developed in Visual C++ using object oriented programming, MuRoS allows the implementation of robots with different controllers, sensors and actuators. New types of robots can be easily implemented inheriting characteristics from the classes already developed. Also, robots of these classes can be directly used in the simulations. The robots developed so far are controlled using the distributed hybrid approach described in the previous chapters: the behavior of each robot is described by a hybrid automaton and robots can switch among different discrete states (modes), having different continuous controllers in each mode. Switches between modes are triggered based on the robots continuous states, perception of the environment and explicit communication. Several controllers have been experimented in the simulator, such as potential fields, path following, leader-follower and open loop approaches. Both implicit and explicit multi-robot communication can be simulated, allowing the robots to exchange information during the task execution. For explicit communication, a message passing system was implemented. Robots also have sensing capabilities and the ability of building maps and plans in real time.

Developed for the MS-Windows system, MuRoS has a graphical user interface that allows the construction of different environments containing a large number of robots and obstacles. The simulator has a very good performance and results are obtained very rapidly. Execution can be observed in real time and several parameters such as controller gains, integration step, etc., can be modified dynamically. Also, results can be exported to other tools such as Matlab for data analysis. Figure 5.1 shows a snapshot of the simulator with 4 nonholonomic robots carrying an object, 10 holonomic robots moving towards the goal, and 3 obstacles. A more detailed description of the simulator is presented in Appendix B.

5.2 Cooperative Manipulation

Our first simulations using the dynamic role assignment mechanism were performed in a cooperative manipulation task, in which a team of robots cooperates to carry a large object in an environment containing static and dynamic obstacles. As mentioned before, cooperative manipulation is one of the most widely used testbeds for cooperative robotics.

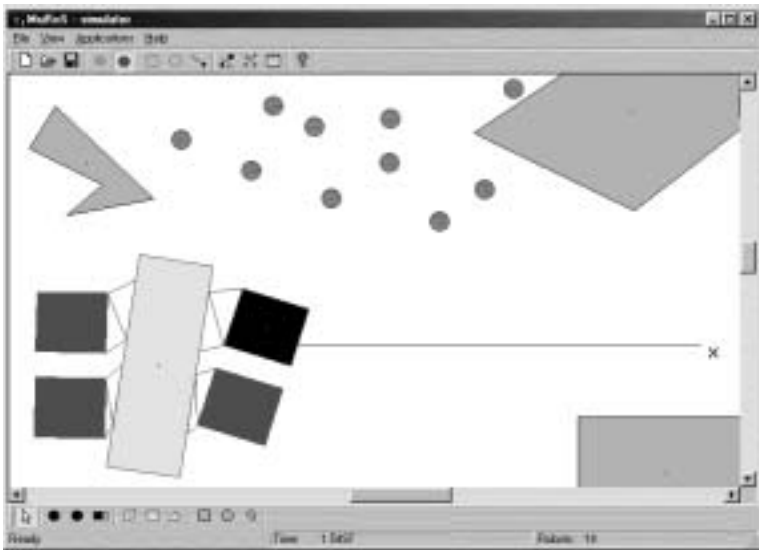


Figure 5.1: Snapshot of MuRoS interface.

It is also a classical example of a tightly coupled task because it cannot be performed by a single robot working alone and requires tight coordination to grasp and transport the object without dropping it. We simulated different scenarios for cooperative manipulation, with different types of robots, coordination hierarchies and communication strategies.

5.2.1 Description

Basically, the approach used for the cooperative manipulation was a leader-follower architecture as discussed in Chapter 3 (Figure 3.1). One robot is identified as a leader, while the others are designated as followers. The assigned leader has a planner and broadcasts its estimated position and velocity to all the followers using data messages. Each follower has its own trajectory controller that acts in order to cooperate with the leader. The planner and the trajectory controllers send set points to the low level controllers that are responsible for the actuators. All robots have a coordination module that controls the cooperative execution of the task. This module receives information from the sensors and exchanges control messages with the other robots. It is responsible for the role assignment

and for other decisions that directly affect the planners and trajectory controllers.

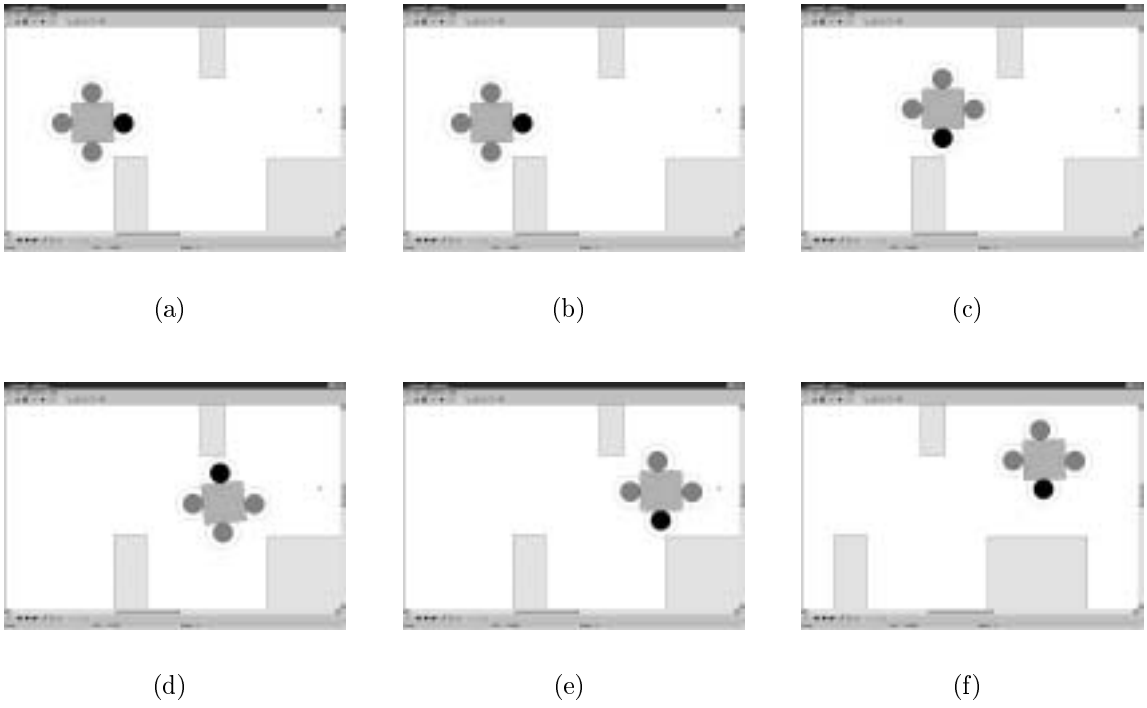
The role assignment is used to exchange the leadership responsibilities among the robots: at any moment, the robot performing the leading role can become a follower, and any follower can take over the leadership of the team. There are basically three roles in this task: Dock, where the robots coordinate themselves to get the object, and Transport, which in fact is composed by the subroles Lead and Follow. The hybrid automaton for this task (controllers and transitions related to these roles) will be detailed in the next chapter when we present the cooperative manipulation using real robots. It is important to mention that the main purpose of the leadership exchange mechanism here is to allow the robots to react and adapt easily to sudden events such as obstacle detection, sensor failures, etc. It is also important to divide the leadership among the robots in such a way that, in each phase of the cooperation, the robot that is best suited in terms of sensor power, manipulation capabilities, etc., will be leading the group.

We implemented two methods for changing the leadership: request and resignation. In the leadership request, one of the followers sends a message requesting the leadership. This normally happens when one of the robots is not able to follow the leader's plan and/or knows a better way to lead the group in that moment. For example, if one of the followers detects an obstacle, it can request the leadership, avoid the obstacle, and then return the leadership to the previous leader. In the resignation process, the leader relinquishes the leadership to another robot. This may happen when the robot senses that it is unable to continue leading or when it finishes its leading turn in a task that has multiple leaders. The leadership can be offered to a specific robot or to all robots simultaneously.

5.2.2 Simulation Results

Initially, two different experiments were performed with the simulator to show the effectiveness of the role exchange in cooperative manipulation [Chaimowicz et al., 2001b]. In the first experiment, four holonomic robots cooperate in order to carry an object from an initial position to the goal. In this experiment, we show the leadership request mechanism:

one of the robots requests the leadership when it senses that it will not be able to follow the path determined by the leader. Figure 5.2 shows snapshots of the simulator during the task execution.



(a) (b) (c)

(d) (e) (f)

Figure 5.2: Leadership request with four holonomic robots transporting an object.

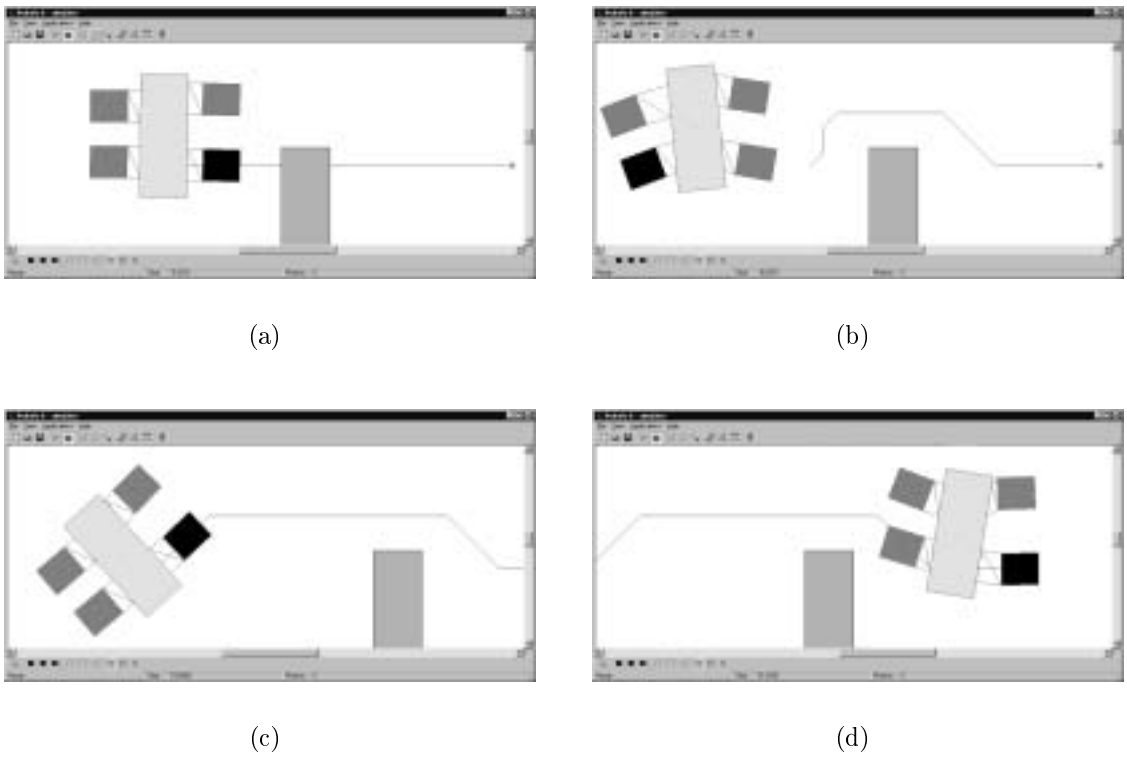
Snapshot (a) in Figure 5.2 shows the robots before starting the docking phase. The robots are represented by circles and the object to be carried is the square in the middle of them. Each robot has a sensing area represented by a dashed circle around it. The other rectangles on the environment are obstacles and the goal is marked with a small x on the right of the figure. After docking, the robots start transporting the object (snapshot (b)). To choose the initial leader, the robots exchange information and the robot that is closer to the goal (robot on the right, shown in black) is elected. As explained in the previous section, the leader has a trajectory to the goal, and continuously sends its state information to the followers, that move in order to cooperate with it. But the current leader is unaware

of the first obstacle, that is outside its sensing region. This obstacle is on the path of one of the followers (the robot on the bottom). When this robot senses the obstacle, it broadcasts a message requesting a role exchange. The leader receives this message and sends another message passing the leadership to the requester. The new leader is then able to avoid the obstacle and continue moving towards the goal, as shown is snapshot (c) in Figure 5.2. The same thing happens with the next two obstacles: the robot on the top assumes the leadership to avoid the second obstacle (snapshot (d)) and the robot on the bottom requests it again to avoid the last obstacle (snapshot (e)) before reaching the goal (snapshot (f)).

The second simulation shows another type of dynamic leadership exchange. The leader senses that it is not able to continue leading and resigns the leadership, that is taken by one of the other robots. Figure 5.3 shows four nonholonomic robots manipulating an object using compliant arms. This type of robot is similar to one of the real robots that will be presented in the next chapter. Snapshot (a) in Figure 5.3 shows the robots starting the manipulation task. The robot on the bottom right is the initial leader and the line between the leader and the goal is the planned path[1]. Note that the robot has not detected the obstacle yet, so it plans a straight line to the goal. When the leader detects the obstacle (snapshot (b)), it replans its path, and senses that it will not be able to continue leading due to its controller constraints. Therefore, it sends a message resigning the leadership. In this message, it includes other information, such as its planned path, the position of the obstacle, etc. This information is used by the other robots to select the new leader. The robot on the bottom left is chosen and performs a backup maneuver in order to help the previous leader. After this maneuver, the robot returns the leadership to the previous leader (snapshot (c)) that replans its path and resumes the transportation avoiding the obstacle (snapshot (d)).

We also simulated leadership resignation considering possible communication failures among the robots. This specific simulation was developed as part of a larger project where

[1]This path was generated using the Grassfire algorithm that will be explained in Appendix B.



(a) (b)

(c) (d)

Figure 5.3: Leadership resignation with four nonholonomic robots transporting an object.

we propose the use of implicit communication for the leadership exchange in a cooperative box carrying task [Pereira et al., 2002]. Two nonholonomic robots equipped with compliant arms were used on the simulations. Figure 5.4 shows a snapshot of the simulation, where the Leader is the black robot and the Follower is the dark gray one. A trace of the box position is also shown. The goal is located at the top-right corner of the screen, inside the dashed rectangle.

The communication models used in this simulation are the following: in the explicit communication, the robots exchange asynchronous messages and failures are simulated by "losing" a percentage of these messages during transmission. An acknowledgment message is sent to confirm that a message was received and, in case of failure, the sender transmits the message again after a timeout period. On the other hand, using implicit communication, the leading robot simply starts to move backwards at a constant velocity during a

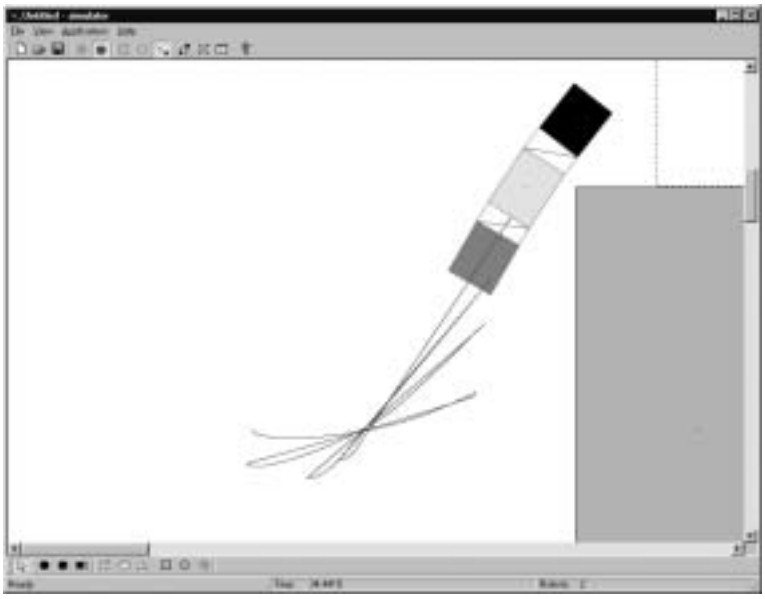


Figure 5.4: Snapshot of the box carrying simulation.

given period of time. The follower senses this movement through the compliant arm, and automatically assumes the leadership.

Thus, there are four possible roles in this task: Dock, where the robots approach the box, Lead, Follow, and Wait. The Wait role is necessary because of the possible failures in communication during the role exchange. After sending a message to change the leadership, the robot should wait for an acknowledgment in order to continue execution. As explained, robots are controlled by different continuous equations in each role. Therefore, it is interesting to observe how these equations evolve with time during the execution of the task. The graphs of Figure 5.5 present the linear velocities of robot 1 and robot 2 as a function of time during task execution (robot 1 is the gray robot in Figure 5.4). Together with the velocities, the graphs present the roles (control modes) of each robot. It is possible to observe how the continuous equations behave in each control mode and the discontinuities due to discrete transitions. For example, the velocity of robot 2 abruptly changes from -100 to 0 when the robot changes from the Lead role to the Wait role (shortly after time = 5). In this case, robot 2 detected an obstacle and stopped, offering the leadership to the other robot and waiting for a response.

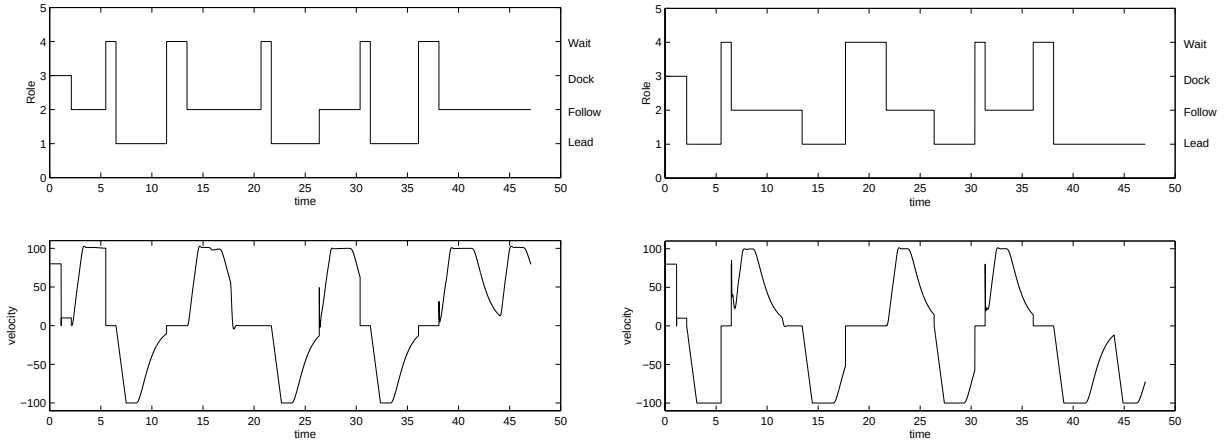


Figure 5.5: Roles and linear velocities of robots 1 (left) and 2 (right) during task execution.

To compare implicit and explicit communication mechanisms, the simulation time spent by the two robots to carry a box from an initial position to the goal was measured. Both implicit and explicit communication were used and communication failures were simulated by increasing the rate of lost messages during execution. Figure 5.6 shows the task completion time using implicit and explicit communication, increasing the rate of failures in the transmission from 0 to 90%. For each point of the graph, 200 runs were executed and the arithmetic mean was computed.

As expected, the execution time increases when communication failures are inserted in the system. The robots use an acknowledgement/timeout mechanism, consequently, the total waiting time will be significant in faulty communication situations. Another important result is that the completion time using implicit communication is constant and similar to the time using explicit message exchange in a reliable environment. In this case, the performances are similar because the explicit communication is used only to pass the leadership. No other additional information that could increase the performance, such as trajectory direction or leadership requests, are transmitted by the leader. Also, failures in the implicit communication that can degrade the task overall performance are not being considered.

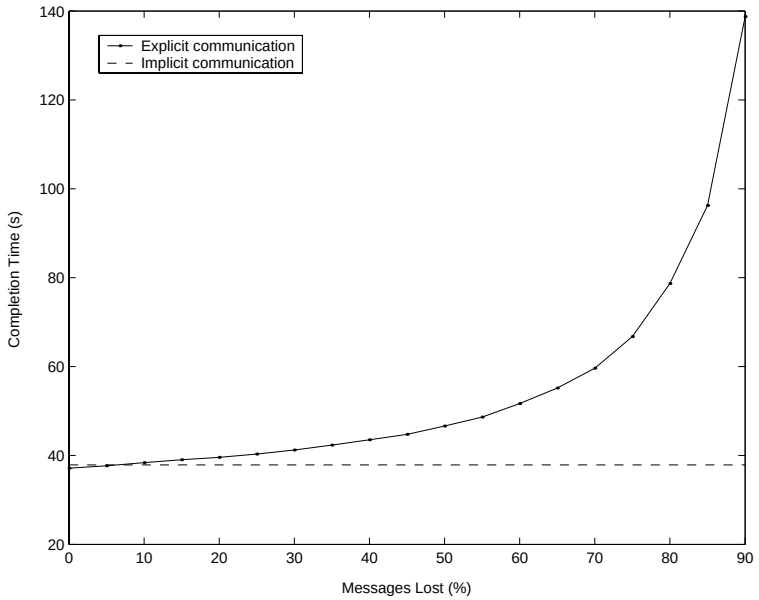


Figure 5.6: Completion times for implicit and explicit communication.

5.3 Cooperative Search and Transportation

In this section, our role assignment mechanism is demonstrated in a Cooperative Search and Transportation task, in which a group of robots must find and cooperatively transport several objects scattered in the environment [Chaimowicz et al., 2002]. It is a combination of a loosely coupled task, where the robots search the area looking independently for objects, and a tightly coupled task, in which the robots must manipulate objects in cooperation. This task is similar to the cooperative search and rescue proposed in [Jennings et al., 1997] with the basic difference that more than one object must be transported to complete the task. Another similar task is the object sorting described in [Lin and Hsu, 1995], where groups of robots must transport several objects between different locations in a bounded area. We use the cooperative search and transportation to demonstrate the role reallocation mechanism, showing that its use can improve the task performance and avoid deadlocks.

5.3.1 Description

The cooperative search and transportation task can be stated as follows: a group of n robots must find m objects that are scattered in an area and transport them to a goal location. Each object i requires at least k robots ($k > 1$) to be transported and has a importance value v. Thus, each object can be described by a pair $\{k, v\}$, representing respectively the amount of work (in terms of number of robots) and the reward for that object. Differently from a common foraging task, in which the robots can act independently from each other and communication is not strictly necessary, this task requires the robots to coordinate themselves in order to transport the objects in cooperation.

The coordination of the robots is done using the role assignment mechanism. Basically, there are five different roles in the cooperative search and transportation. In each of these roles, the robots compute a utility μ that is used in the role assignment mechanism. Initially, all robots start in the Explore mode, in which they randomly move in the environment searching for items to be transported. When a robot detects an object inside its sensing area, it finishes its exploration role and starts the Attach role, in which it approaches the object preparing to transport it. If a robot is the first one to attach to an specific object, it assumes the Attach Lead role. Besides approaching the object, the attach leader is responsible for broadcasting messages informing the other robots about the new role available, and the number of volunteers that are necessary to transport that object to the goal. All robots that receive this message compare the new role utility $\mu_{r'}$ with their current utility μ_r and send a message back to the attach leader if they want to volunteer for the new role. This works like a bidding process, where the volunteers with the higher utility values are recruited by the attach leader. These robots reallocate to the Approach role and start moving towards the object. When the object is inside the robot's sensor range, it assumes the Attach role. When the number of robots necessary to carry the object is sufficient, they assume the Transport role and cooperatively move the object to the goal.

When a robot assumes the Approach or Attach roles, it makes a commitment to the

attach leader. The attach leader keeps broadcasting messages in a fixed rate offering the role until the number of committed robots is sufficient to transport the object. If a committed robot reallocates to another role, it must send a message to the leader resigning the previous role. It is important to mention that we are not considering robot and communication failures in these experiments.

In each one of these roles, robots may be controlled by different continuous equations. For example, in the Explore mode they move randomly while in the Approach and Attach modes they use a potential field like controller in order to approach the objects. Also, other continuous and discrete variables may be stored within each mode and updated during the execution of the task. Figure 5.7 shows the hybrid automaton for the cooperative search and transportation. For clarity, only the control modes (roles) and discrete transitions (role assignments) are presented. The solid arrows represent the role allocation and the dashed arrows represent the reallocation, in which the robots interrupt the performance of one role to assume another. As mentioned in Chapter 3, a robot performing a role r reallocates to another role r' when the difference $\mu_{r'} - \mu_r$ is greater than a threshold τ. There are four role reallocations in this diagram: the first one is when an explorer volunteers and is recruited to approach a certain object, as explained before. The same thing can happen when the robot is already in the Attach mode and an Approach role with better utility is offered. The other two reallocations happen from/to the Attach Lead mode: an attach leader can reallocate itself to an Approach role with higher utility if its object has no other attachers. In this case, the robot stores its position in a local memory in order to return to this object after finishing the new role. Also, a robot that is approaching can become a attach leader if it finds a new object and the utility of the new role is higher than its current utility. Another kind of reallocation that is possible is when a robot approaching an object i reallocates to approach a different object j. In this case, the robot will be performing the same role but with a different parameter.

The choice of a suitable function to measure the role utilities is an important aspect of the task. The execution of the role assignment mechanism and consequently the performance of the task will vary according to the function chosen to measure the role utilities.

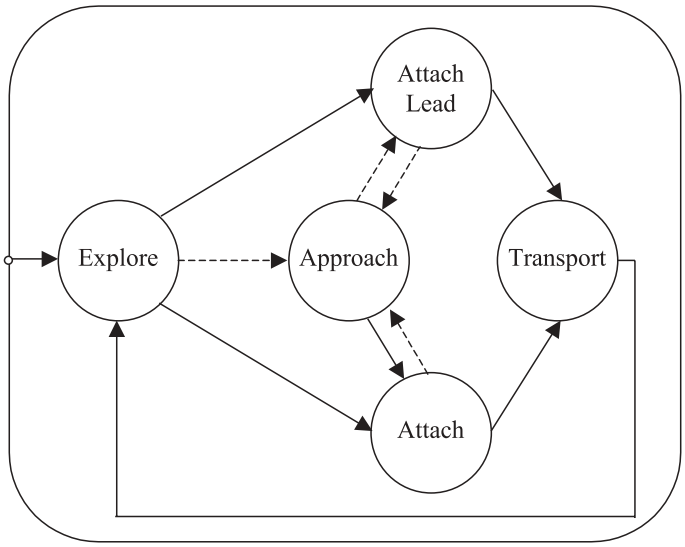


Figure 5.7: Control modes (roles) and discrete transitions (role assignments) for the cooperative search and transportation.

Depending on the objective of the cooperative search and transportation, for example minimize execution time or maximize the value in a shorter time, different utility functions can be implemented. In the experiments presented in this thesis we use a simple heuristic function in order to test the execution of the role assignment mechanism. We do not intend to compare different functions, analyze performance in detail, or search for optimal results. Instead, we just want to provide a simple testbed for our role assignment mechanism. The selection of optimal utility functions for the role assignment (and for task allocation in general) is a difficult problem in itself and is beyond the scope of this thesis. In Chapter 7 we outline some future work that can be done in this area.

The utility function μ used in the experiments presented here is defined as follows: robots performing the Explore role have a very low utility (0) while robots transporting an object have the highest utility (∞). For the other roles, we have defined an utility function that balances the value of the object (v) with the number of robots being waited to start the transportation (k_w) and a function of the distance to the object ($f(d)$). Thus,

the utility of performing a role r is given by:

$$\mu = \begin{cases} 0, & r = \text{Explore}, \\ \infty, & r = \text{Transport}, \\ \frac{v^2}{k_w+1} + \frac{1}{f(d)}, & otherwise. \end{cases}$$

Using this heuristic function, each robot tries to maximize the value recovered in a short time but also gives priority to objects that need fewer robots to be transported and are near the robot's current position. Note that robots in the Transport mode will never be reallocated while robots performing the Explore role have a great probability of being reallocated, depending on the threshold. For example, for a threshold $\tau = 0$ the robots in the Explore mode will always be reallocated.

5.3.2 Simulation Results

The dynamic role assignment in the cooperative search and transportation task was implemented and tested using MuRoS. Figure 5.8 shows a snapshot of the simulator during the execution of the task. In this figure, the goal is represented by a square area marked with an x and the objects are represented by the five circles with numbers (pairs $\{k, v\}$) inside. Two of them (inside the goal region) have already been transported. The small circles are the robots and the dashed circles represent the boundary of their sensing area. The robot color represents its current role: two white robots, one at the bottom right and the other at the top left of the screen, are in the Explore mode. Three black robots at the center of the screen are transporting the object marked with the numbers $\{3, 1\}$ to the goal. At the bottom left, there is the attach leader (light gray) and two gray robots attaching the object $\{5, 1\}$. The other two robots (dark gray) are approaching the same object.

The experiments were performed using 20 holonomic robots and 30 objects randomly distributed in the environment. The dimensions of the search area and the goal region are 30 by 30 meters and 4 by 4 meters respectively. Comparatively, the diameter of each

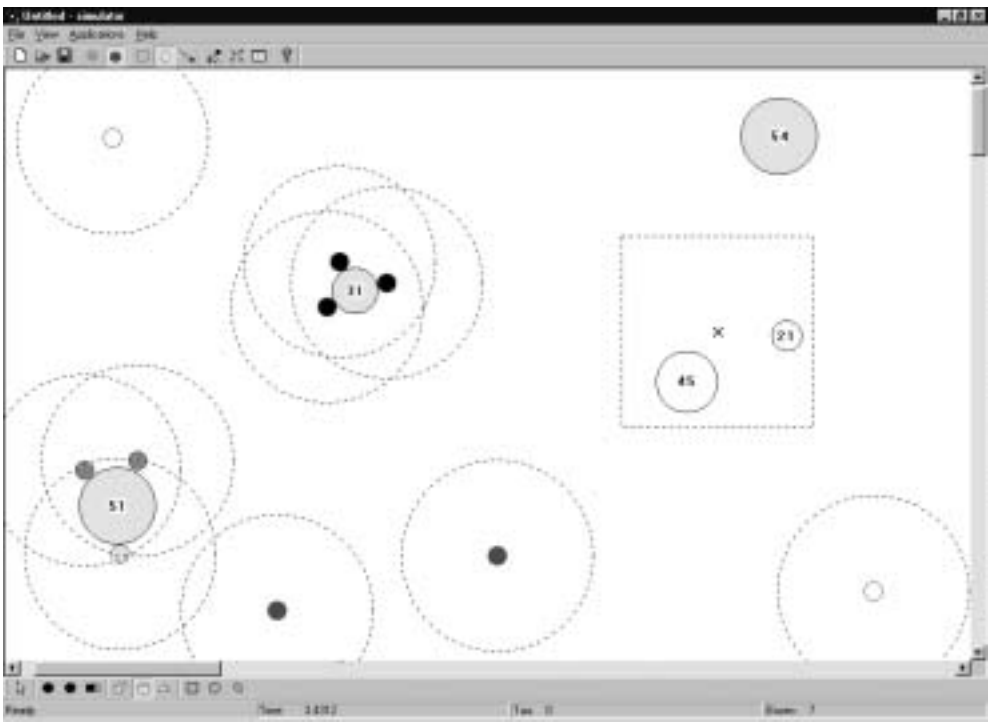


Figure 5.8: Snapshot of the simulator during the execution of the cooperative search and transportation task.

robot is 30 centimeters. The value of each object (v) and the number of robots necessary to transport it (k) were generated randomly, with $v = \{1, \ldots, 5\}$ and $k = \{2, \ldots, 5\}$. The utility function described in the previous section was utilized varying the threshold τ from 0 to 8. For each value of τ, 100 runs were performed and the mean values were computed.

Firstly, the time to complete the task was measured. The results are presented in Figure 5.9. The graph shows that the completion time starts to increase for values of τ greater than 2. This result was expected because the number of role reallocations decreases as τ increases. With few reallocations, the robots act more independently as they do not accept new role offers. In this situation, the work force is divided and the level of cooperation decreases since each robot only attaches to the objects detected by itself, not accepting offers from its teammates. Consequently, the time to gather the k robots necessary to transport each object will increase, also increasing the overall time to complete the task.

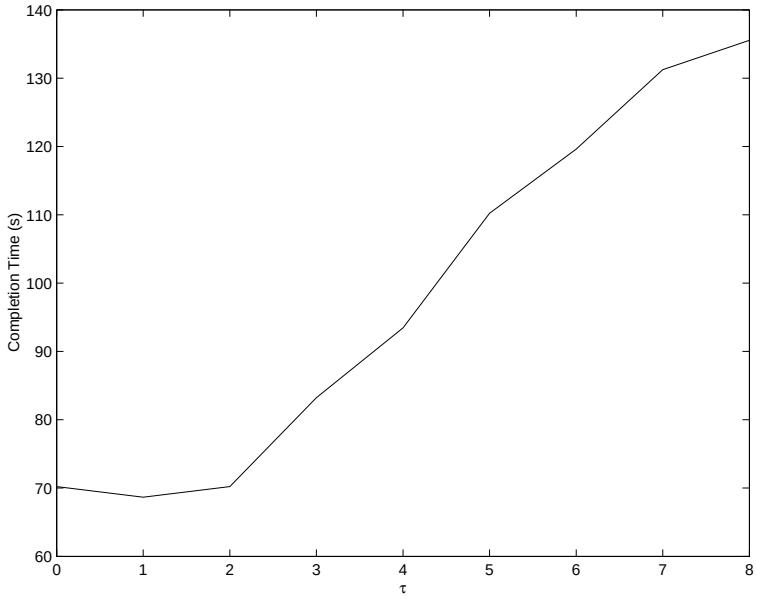


Figure 5.9: Completion time varying the threshold τ.

The extreme case of this work division causes deadlocks. A deadlock occurs when all robots are performing the Attach role to specific objects, but the number of robots attached to each object is not sufficient to transport it. In this case, the robots keep waiting indefinitely and do not complete the task. Figure 5.10 shows the number of deadlocks for each value of τ. More than 50% of the runs with large values of τ results in deadlocks.

In this task, deadlocks are detected by a timeout period, *i.e.*, if the robots do not complete the task within a certain amount of time then a deadlock has occurred. The timeout mechanism works fine in this situation, where the only cause for not completing the task is the presence of a deadlock. But more elaborated mechanisms to detect deadlocks can be used in general situations. An example is the use of a superimposing process that monitors the global state of the system and detects if it is changing as time flows. In the cooperative search and transportation, the global state can be a set composed by the current role plus the position of each robot. If there are no role assignments and the robots do not move within a fixed period of time, then the global state is not modified and the system is deadlocked.

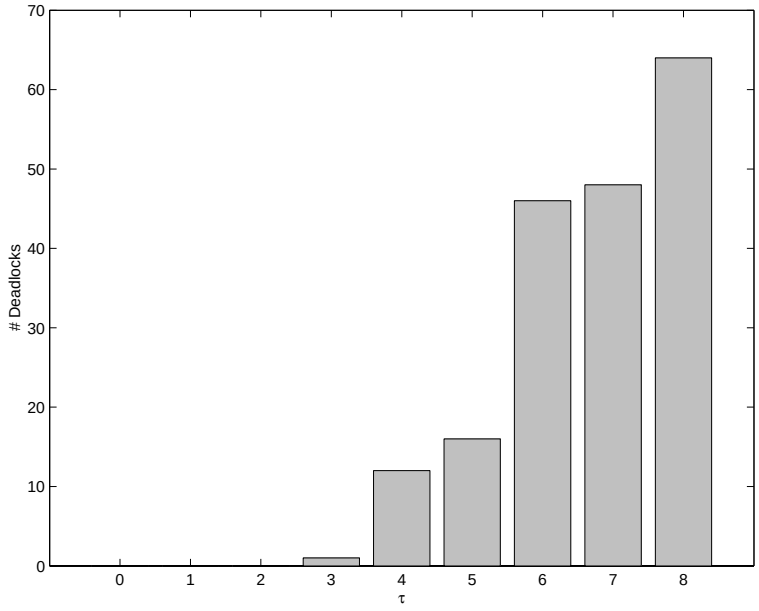


Figure 5.10: Number of deadlocks (in 100 runs) for different values of τ.

Another result that can be observed is that the use of dynamic role assignment with the utility function explained above helps to maximize the total value transported in the beginning of the execution. The graph of Figure 5.11 shows the percentage of the total value recovered as a function of the execution time for different values of τ. It can be observed that for small values of τ, objects with larger values are transported first, according to the utility function that prioritizes objects with large values (v^2). But it is important to mention that depending on the main objectives of the task, this utility function may not be adequate. For example, if the main objective is to decrease the distance covered by the robots, the results obtained with this utility function are not satisfactory. A small threshold causes more reallocations making the robots move from one object to another. This implies in a larger distance covered by each team member.

Observing the results presented in this section, it can be seen that the dynamic role assignment allows the successful execution of the cooperative search and transportation task, demonstrating that this mechanism can be used for the coordination of multiple

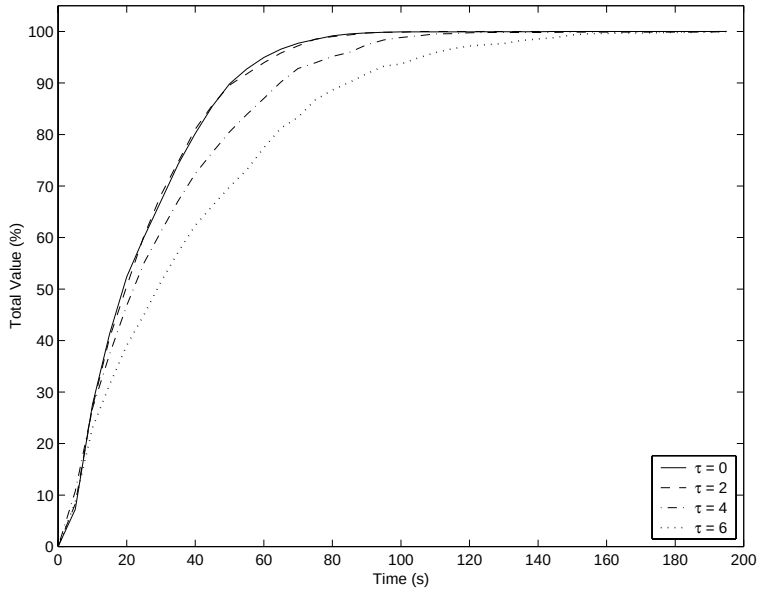


Figure 5.11: Percentage of the total value transported as a function of execution time.

robots. In this specific case, the use of the role assignment with a suitable utility function and adequate threshold values brought good performance in terms of time and other metrics while avoiding deadlocks that would prevent the task completion.

Chapter 6

Real Experiments

In this chapter, we present some of the real experiments performed to show the effectiveness of the dynamic role assignment and hybrid systems modeling of cooperative robotics. In these experiments, two and three heterogeneous robots work cooperatively to transport a large box between two locations in a environment that may contain static and dynamic obstacles [Chaimowicz et al., 2001c].

Here, in the same way as in the simulations, we used a leader-follower approach in which the leader broadcasts its position and velocities and the followers have independent trajectory controllers and act in order to collaborate with the leader. The role assignment is used to exchange the leadership among robots: the leader can resign it or one of the followers can request the leadership. This allows the robots to divide responsibilities and adapt to unexpected situations that can occur during task execution.

6.1 The Team of Robots

Three heterogeneous robots with different sensing capabilities, driving mechanisms and operating systems were used in the experiments. Figure 6.1 shows a picture of the robots.

The robot on the left in Figure 6.1 is a TRC Labmate platform, equipped with an actively controlled compliant arm [Sugar and Kumar, 1998b]. The platform is non-holonomic,

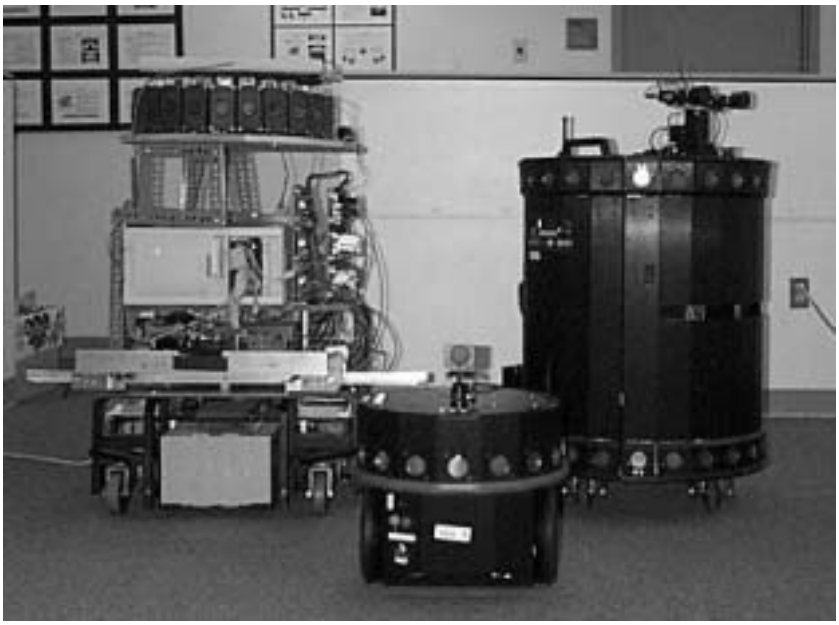


Figure 6.1: Team of heterogeneous robots.

and the only on-board sensors are encoders located at the arm and at the two actuated wheels. All the programming is done using Simulink and Real Time Workshop (RTW) and compiled for DOS. Simulink is a very powerful tool for the simulation of continuous systems. Together with Real Time Workshop, it can be used to generate code for real applications. For this, it is necessary to implement specific blocks of code to integrate the Simulink model with real devices such as wireless communication boards and robot motors.

The robot on the right in Figure 6.1 is a XR4000, developed by Nomadic Technologies. It has a holonomic driving system offering three degrees of freedom and has several sensors, including two rings of 24 ultrasound and infrared sensors, a stereo vision system, and encoders. It is also equipped with a fork-lift arm that has one prismatic joint along the vertical axis. It uses the Linux system and the programming is done using C. The third robot is a Nomad Super Scout II. In the same way as the XR4000, it has a Pentium-based PC processor running Linux, but it has a non-holonomic differential driving system. The Scout is equipped with ultrasound and contact sensors, but it does not have manipulation capabilities.

All robots are equipped with wireless Ethernet boards and exchange messages using the IPX protocol. IPX is a connectionless datagram protocol. The robots do not have to establish a connection to exchange packages and each package is treated as an individual entity, having no logical or sequential relation to another package. Consequently, IPX packages are addressed and sent to their destination, but there is no guarantee or verification of successful delivery. Because of these characteristics, IPX is a fast and simple communication protocol being suitable for this kind of application. Data messages are continuously broadcast by the leader at a frequency of 20Hz and control messages are sent asynchronously, upon necessity. Because IPX does not provide delivery confirmation, an acknowledgment mechanism was developed. This is not necessary for data messages, that are continuously sent, but is important for control messages, that are used in the task control. All messages are broadcast using specific sockets and received by all the robots. Different sockets are used for data and control messages and there is no one-to-one communication. If a message needs to be specific to one robot, it is possible to put its identification number in the package header, and it will only be considered by that robot.

6.2 Modes and Controllers

Different controllers and planners are used by each robot depending on its role in the task. The two robots carrying a box must tightly coordinate themselves to keep the balance of the box while they are moving to the desired position. When the third robot is used as a remote sensor, it must maintain a certain position with respect to the other robots in order to detect possible obstacles.

To determine the kinematic equations and inputs it is necessary to consider the characteristics of each robot in each mode. In this cooperative manipulation task, the robots can be in one of the following control modes: Dock, where they must coordinate themselves to grasp and pick up the box, and Transport, where they march in a coordinated fashion. Figure 6.2 presents these modes using behavioral hierarchy diagrams. Diagram (a) shows the high level modes of the task. The Transport mode consists of two submodes Lead, and

Follow, as shown in Diagram (b). Diagram (c) presents the submodes of the XR4000's Dock mode. In this task, the Transport mode is similar for all robots, but the Dock mode depends on the sensing and manipulation capabilities of each one.

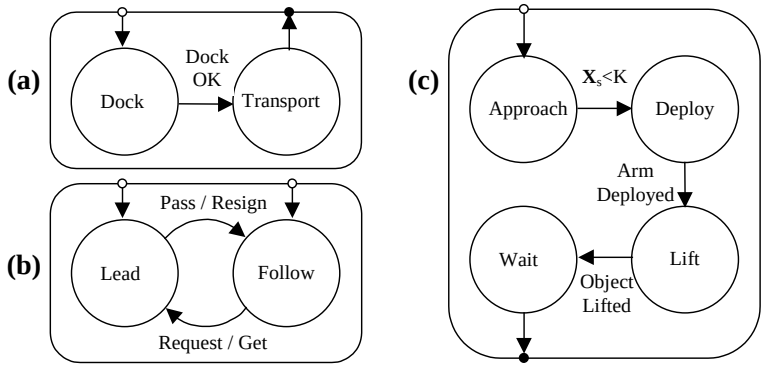


Figure 6.2: Behavioral hierarchy diagrams. (a) High level modes, (b) Transport mode, and (c) XR4000's Dock mode.

Figure 6.3 shows a diagram of the two robots carrying a box in an environment with obstacles. Since the Labmate is non-holonomic the inputs for its low level controllers are the linear and angular velocities ($u_1 = v$, $u_2 = \omega$). Thus, the state equations become:

$$\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u}) \rightarrow \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}.$$

The input will depend on the current mode of the robot. In the Dock mode the inputs are computed based on the state of the compliant arm ($\boldsymbol{x}_a$), shown in Figure 6.3:

$$\text{Dock} \begin{cases} u_1 = f(\boldsymbol{x}_a), \\ u_2 = g(\boldsymbol{x}_a). \end{cases}$$

The behavioral hierarchy diagram for the Transport mode is shown in Figure 6.2(b). It consists of two submodes: in the Lead mode, the Labmate uses an open loop planner, and in the Follow mode, it uses information sent by the leader together with feedback

information from the compliant arm to compute its input. On our notation, the superscript ($\hat{}$) represents estimates of the state of the other robots. If we use the subscript l to refer to the leader, and D is the distance between the robots, the control laws for the submodes are given by the following equations:

$$\text{Lead} \begin{cases} u_1 = v(t), \\ u_2 = \omega(t), \end{cases} \qquad \text{Follow} \begin{cases} u_1 = \hat{v}_l \, cos(\hat{\theta}_l - \theta) + f(\boldsymbol{x}_a), \\ u_2 = (\hat{v}_l / D) \, sin(\hat{\theta}_l - \theta) + g(\boldsymbol{x}_a). \end{cases}$$

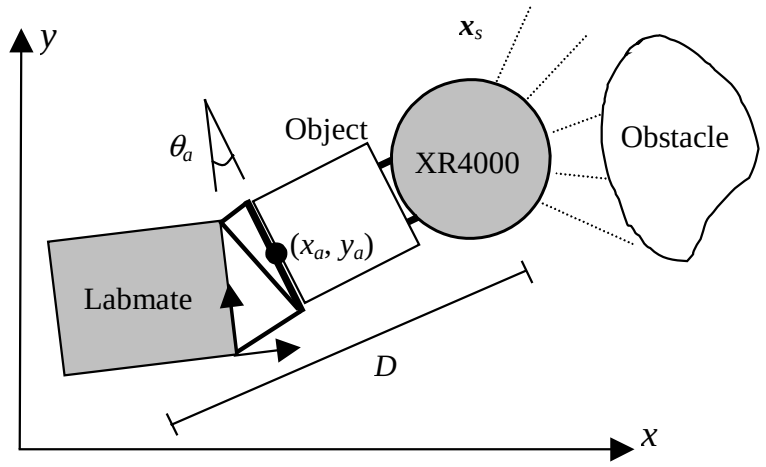


Figure 6.3: Diagram of the Labmate and the XR4000 carrying an box. The states of Labmate's compliant arm ($\boldsymbol{x}_a = [x_a, y_a, \theta_a]^T$) and of XR4000's infrared sensors ($\boldsymbol{x}_s$) are shown.

The XR4000 is holonomic, having three degrees of freedom and, consequently, three inputs ($\dot{x} = u_1$, $\dot{y} = u_2$, $\dot{\theta} = u_3$). The behavioral hierarchy diagram for the Dock mode is shown in Figure 6.2(c). Note that there are several submodes in the XR4000. The information from the infrared sensors ($\boldsymbol{x}_s$) is used by the XR4000 in the Approach submode:

$$\text{Dock (Approach)} \begin{cases} u_1 = f(\boldsymbol{x}_s), \\ u_2 = g(\boldsymbol{x}_s), \\ u_3 = h(\boldsymbol{x}_s). \end{cases}$$

The Transport mode is similar to those in other robots (see Figure 6.2(b)), but the continuous equations of the submodes are different. The Lead mode uses a planner and the Follow mode uses a simple proportional controller based on the information sent by the leader. The terms x^d, y^d, θ^d are set points that depend on the task. For example, when the Labmate is leading the task we have: $x^d = \hat{x}_l + D\cos\theta$, $y^d = \hat{y}_l + D\sin\theta$ and $\theta^d = \hat{\theta}_l$.

$$\text{Lead} \begin{cases} u_1 = v_x(t), \\ u_2 = v_y(t), \\ u_3 = \omega(t), \end{cases} \qquad \text{Follow} \begin{cases} u_1 = k_1(x^d - x) + f(\hat{\boldsymbol{x}}_a), \\ u_2 = k_2(y^d - y) + g(\hat{\boldsymbol{x}}_a), \\ u_3 = k_3(\theta^d - \theta) + h(\hat{\boldsymbol{x}}_a). \end{cases}$$

The Scout is non-holonomic, having the same state equations as the Labmate, but it is not used directly in the manipulation of the box. Consequently, it does not have a Dock mode and the controller in its Transport mode can be more flexible because its position relative to the leader can vary during the task execution. In the experiment presented in this paper, the Scout uses a planner when leading and sets its velocity to be equal to the leader's velocity in the Follow mode:

$$\text{Lead} \begin{cases} u_1 = v(t), \\ u_2 = \omega(t), \end{cases} \qquad \text{Follow} \begin{cases} u_1 = \hat{v}_l, \\ u_2 = \hat{\omega}_l. \end{cases}$$

6.3 Results

Three different experiments are presented to demonstrate the dynamic leadership exchange with real robots. In all experiments, the XR4000 and the Labmate cooperate to carry a box (Figure 6.4) between two different locations in the environment, but different scenarios in each experiment demonstrate several features of the architecture. The graphs of figures 6.5, 6.6, and 6.9 show the trajectories executed by the robots in each experiment with data acquired from odometry. The numbers inside the graphs indicate the initial positions and the points where the leadership has been changed. Each robot is indicated by a letter (X–XR4000, L–Labmate, S–Scout).

Before beginning the transportation, the two robots must coordinate themselves to get

the box. This is called the Dock mode of the cooperation. The XR4000 uses its infrared to approach the box and deploy its arm at the correct position. The Labmate uses feedback information from the compliant arm to hold the box. Docking is done in one dimension (the robots and the box are aligned), and the Labmate waits until the XR4000 finishes before starting its own docking.

Figure 6.4: Two robots carrying a box.

The first experiment, shown in Figure 6.5, demonstrates the leadership resignation mechanism. The XR4000 begins leading (0X), followed by the Labmate (0L), until it detects an obstacle using its infrared sensor (1X). Then it sends a control message resigning the leadership to the Labmate. The new leader moves backwards in a curvilinear trajectory (from 1L to 2L), returning the leadership to the XR4000 (2X) when it finishes its plan. This experiment shows that, instead of trying to avoid the obstacle locally, which is difficult to accomplish while carrying a box in cooperation, the robots can exchange roles: the XR4000 offers the leadership to the Labmate, which takes it and modifies the trajectory. In this case, the modification is a simple open loop reversal with a turn. Note that during the execution the modes and controllers are changed dynamically due to the role exchange.

The second experiment, shown in Figure 6.6, demonstrates the leadership request process. The Labmate begins leading by going backwards in a curvilinear trajectory (from 0L to 1L). The XR4000 begins following (0X) using its controller and requests the leadership

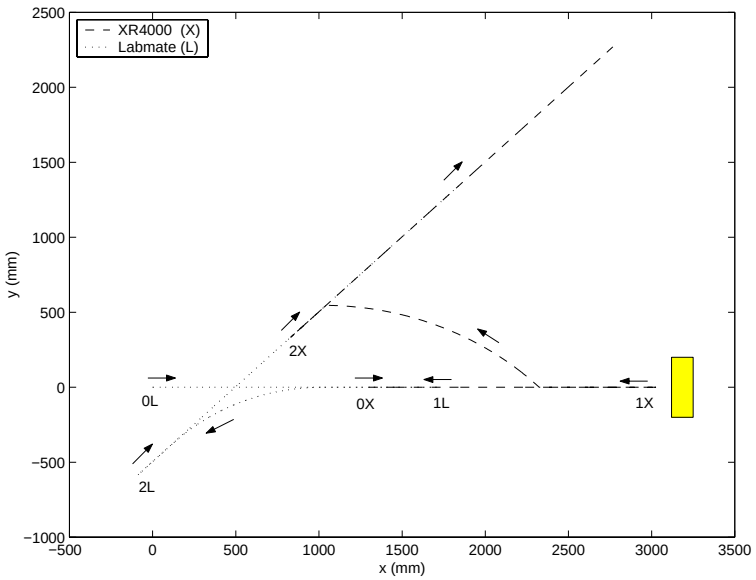


Figure 6.5: Experiment 1 - the XR4000 resigns the leadership (1X) and receives it back (2X).

when its infrared sensor detects an obstacle in its way (1X). After moving to avoid the obstacle, the XR4000 (2X) returns the leadership to the Labmate (2L) that leads until the end of the task. The leadership exchange here is very important because the leader is not aware of the obstacle in the path of the follower. The XR4000 therefore requests the leadership, avoids the obstacle, and returns the leadership to the Labmate.

It is important to note that these experiments are not meant to show or suggest performance of algorithms. In fact, Experiments 1 and 2 show different approaches to how a robot might react in the presence of obstacles.

As mentioned, communication is very important for both coordination and control. Figure 6.7 shows the hybrid automata of the two robots together with messages exchanged by the robots in each role. The circles represent the discrete states of the robots and the rectangles indicate the continuous dynamics (controllers and planners) for each state. The arrows between the two automata are the control and data messages exchanged and the links among the states in a robot are state transitions. To simplify the diagram both the

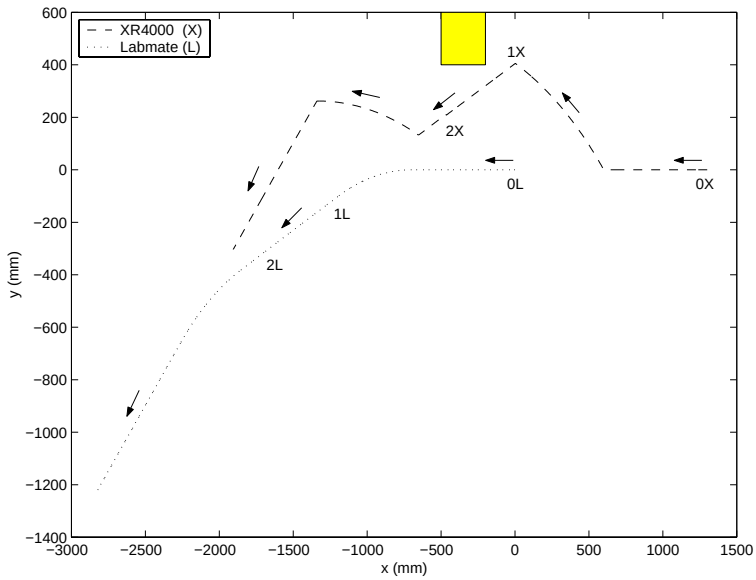


Figure 6.6: Experiment 2 - the XR4000 requests the leadership (1X) and returns it (2X) to the Labmate.

acknowledgment messages and the final states are not shown.

The diagrams for the other experiments are very similar to this one, meaning that it is possible to perform different actions by simply changing the role assignment sequence and the control message flow. It is important to note that, in the experiments presented here, state transitions have a strong relationship with control messages. Transitions normally happen when some event is detected by one of the robots. This robot must notify its teammates using control messages. The reception of a control message by one robot normally causes a state transition because this robot must adjust itself to the changes in the cooperative task. Using the hybrid systems terminology, the transition guards depend on discrete variables that are changed by the reception of some messages. Thus, a transition is taken when a message is received.

Figure 6.8 shows the discrete states (modes) of each robot as a function of time during the execution of Experiment 2. As shown in Figure 6.2(c), the Dock mode of the XR4000 can be divided in four submodes: Approach (A), Deploy (D), Lift (L) and Wait (W). The closeup view in Figure 6.8 shows a detail of a state transition. State transitions do

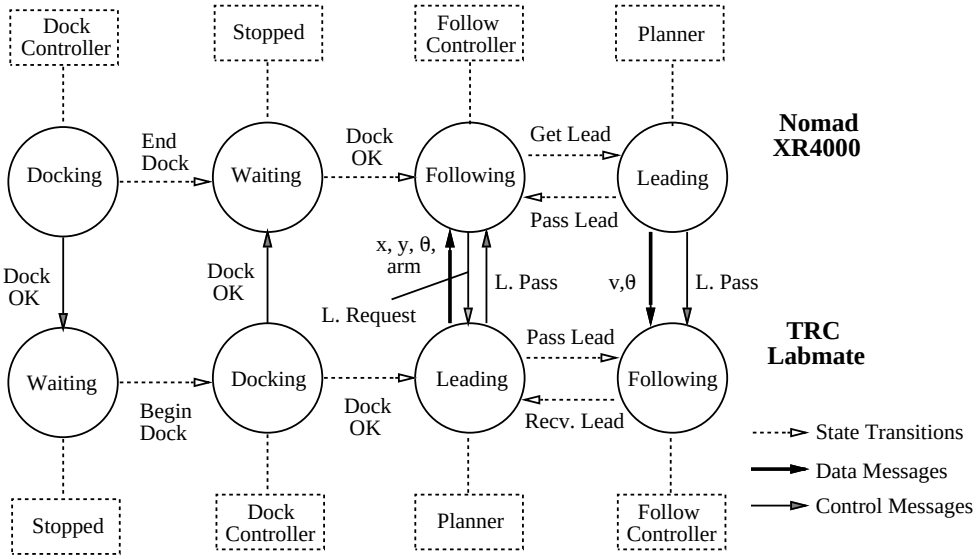


Figure 6.7: The two automata and the messages exchanged in Experiment 2.

not occur simultaneously in the two robots: there is a small delay due to communication. In this specific case, at time t_a the XR4000 sends a message requesting the leadership. The Labmate receives the message at time t_b, changes its state and sends a message to the XR4000 passing the leadership. The XR4000 only changes its state when this confirmation message arrives at time t_c. The interval $t_c - t_b$ is equal to approximately 0.03 seconds and does not affect the execution of the leadership change mechanism.

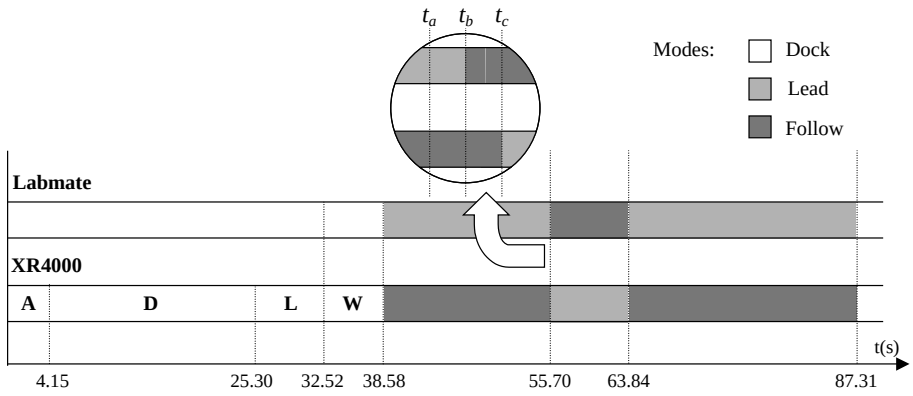


Figure 6.8: Time chart for Experiment 2, with a closeup view of a state transition.

In the third experiment, shown in the graph of Figure 6.9, the Scout is used in the cooperation as a remote sensor for the Labmate. At the start position, the robots are aligned: the XR4000 is in the front (0X), the Labmate in the middle (0L) and the Scout in the back (0S). After docking, the Labmate starts leading, moving backwards until the Scout detects an obstacle using its sonar (1S). Then it requests the leadership, moves to the front (2S) and returns the leadership to the Labmate (2L), who finishes the task making a curve to avoid the obstacle. An interesting fact in this experiment is that the dispatch frequency of data messages had to be reduced to 8Hz, because the Scout was not able to process all the messages using the original frequency of 20Hz. In spite of that, the task was executed without any problem, showing that the architecture can adapt to small changes in communication bandwidth.

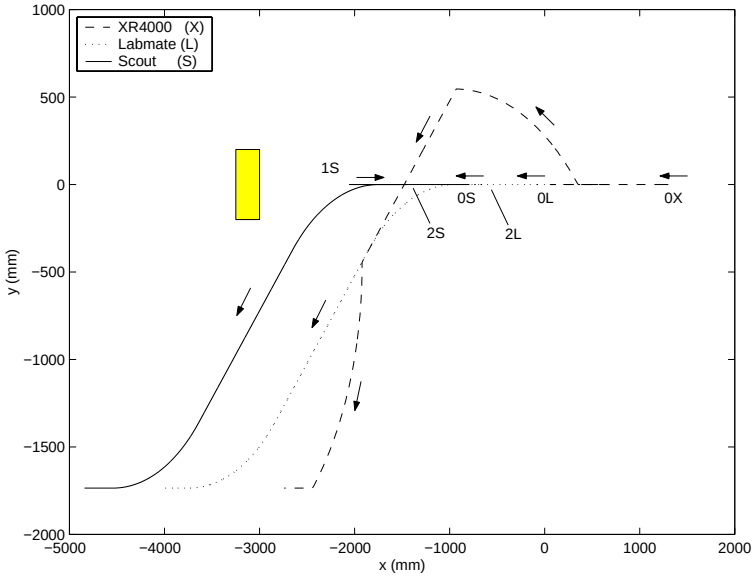


Figure 6.9: Experiment 3 - the Scout requests the leadership in a three robot team (1S) and returns it (2S) to the Labmate.

It is important to mention that, sometimes, the leadership exchange protocol may lead to conflicts that need to be resolved. Two examples in the small team with one leader are: (a) a robot requests leadership but the leader does not relinquish it; and (b) a robot resigns

its leadership, but there are no takers. A related problem is the possibility of a chattering phenomenon where changes in leadership occur too frequently. A priority-based approach is necessary to resolve such conflicts. In the experiments presented in this chapter, we detect deadlocks (using timeout mechanisms, for example) and, in such situations, the command is relinquished to the human operator whose authority supersedes other robots.

Chapter 7

Conclusions

In this chapter we present a summary of the work, highlighting the main contributions of this thesis. We also discuss two important aspects of our framework for cooperative robotic systems, namely applicability and scalability, and we finish the chapter presenting future directions for this work.

7.1 Summary of the Accomplished Work

In this thesis, the problem of how to model and coordinate multiple robots in the execution of cooperative tasks has been addressed. Basically, this work offers two main contributions to the state-of-the-art:

- We proposed a methodology that allows the coordination of multiple robots in the execution of cooperative tasks. This methodology is based on a dynamic role assignment mechanism, in which robots dynamically assume and exchange roles during the task in a synchronized manner, being able to adapt to sudden events in the environment and perform cooperative tasks robustly and efficiently.

- We modeled multi-robot cooperation under a hybrid systems framework, using hybrid automata to specify the behavior of each robot and the parallel composition of automata to model the cooperative task as a whole. This has allowed us to better

formalize the execution of cooperative tasks by multi-robot teams and has provided a framework for developing formal proofs about the cooperation.

The role assignment mechanism and the hybrid systems modeling were used in different cooperative tasks, both in simulations and in real experiments. To perform the simulations, we have implemented MuRoS, a software tool specially developed for the simulation of multi-robot teams in the execution of cooperative tasks. Real experiments were performed in a cooperative manipulation task using two and three heterogeneous robots. The obtained results were very satisfactory, showing that both the role assignment and the hybrid systems modeling are suitable for cooperative robotics.

7.2 Applicability and Scalability

In spite of the good results obtained for different tasks both in simulations and real experiments, the applicability and scalability of this framework are two important issues that should be discussed. The applicability concept refers to the capacity of using this framework in several different cooperative tasks, while the scalability studies if the framework is still valid when the number of robots increases, *i.e.*, if the framework scales well with the number of robots.

We tried to make the role assignment mechanism as general as possible, so that it can be used in completely different cooperative tasks. Basically, a role can execute any kind of action, from a purely reactive controller (sensor/actuator pair) to a more elaborate planning strategy. The role assignment can be performed according to several conditions and different types of information can be stored and transmitted. The same is valid for modeling: hybrid systems is a very powerful framework that can be used to model and study several types of mixed discrete and continuous systems. Different controllers and variables can be defined for each control mode and guards and reset statements in the discrete transitions can be used to model the execution requirements. The only drawback on the applicability of the hybrid systems modeling is that complex systems require the

use of more complex automata to be modeled. This imposes some difficulties in the use of automatic tools for verification, that, as mentioned in Chapter 4, are only available for simpler classes of automata.

In this thesis, we have shown the applicability of our framework in some different tasks such as cooperative manipulation and cooperative search and transportation. In spite of their particular characteristics, other cooperative tasks found in the main journals and conferences (see [Parker, 2000] and [Cao et al., 1997] for an overview) have similar needs for coordination and modeling. These needs are fulfilled in our framework as explained above. Therefore, to the best of our knowledge, the role assignment mechanism and the hybrid systems modeling presented in this thesis may be used successfully in the execution of other cooperative tasks.

Regarding scalability, one of the main factors that should be considered is communication. As shown, depending on the task, the role assignment mechanism should use explicit communication in order to exchange information and synchronize execution. As the number of robots increases, the explicit communication may become a bottleneck because of the large number of messages being exchanged among the robots. At any time, there will be a large number of packages being transported in the wireless network and this can cause delays and failures affecting the task execution. This is even worse if each robot has to communicate with all other robots and this communication should be performed very frequently. There are some solutions that can help minimize the scalability problems related to explicit communication: first of all, as mentioned in Chapter 3, we can impose some kind of hierarchy, dividing the robots in groups and allowing communication only among robots of the same group and among group leaders. Also, we may reduce the frequency in which messages are exchanged and rely more on implicit communication, using sensors to infer information from other robots. These changes will certainly help reduce the number of messages. Besides that, in recent years technological advances have allowed the development of more reliable, affordable, large bandwidth wireless networks, reducing the impact of communication on the scalability of cooperative architectures.

From the modeling perspective, other factor that influences the scalability of our frame-

work is related to the exponential explosion of control modes. In the composition of the hybrid automata of several robots, the number of control modes of the compound automaton increases exponentially with the number of robots. Basically, if we have n robots with m control modes each, the number of control modes in the compound automaton will be limited by m^n. This huge number of states may cause some difficulties in the use of automatic tools for formal analysis, that in general are computer expensive and have a complexity that depends on the number of states. To minimize these problems we may try to reduce the total number of states to be analyzed in the compound automaton. For example, in the compound automaton of Section 4.6, we focused only on the control modes that are reachable from the initial state.

In spite of these possible drawbacks regarding scalability, we have been working together with researchers from the GRASP Laboratory [Song and Kumar, 2002] to simulate a large number of robots in the execution of cooperative tasks. Figure 7.1, for example, shows a cooperative manipulation task being executed in MuRoS by 50 robots.

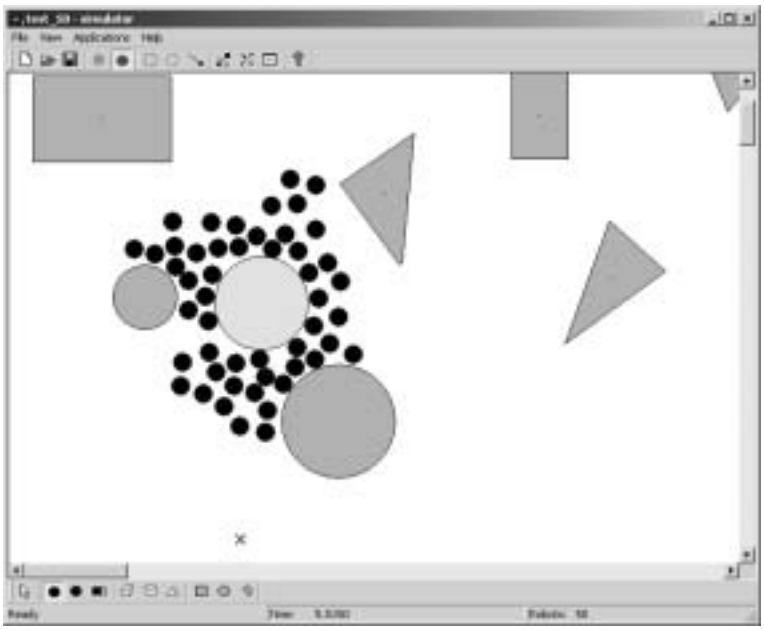


Figure 7.1: Scalability: 50 robots executing the cooperative manipulation task.

7.3 Future Work

There are several possibilities of research in order to continue the work developed in this thesis. The first is to continue the research of hybrid systems modeling of cooperative robotics. As discussed in this thesis, we believe that hybrid systems are well suited to represent and study multi-robot teams in the execution of cooperative tasks. But there are important points that should still be addressed, for example, how to use the hybrid modeling in order to obtain formal proofs about a cooperative task execution. As mentioned in Chapter 4, hybrid systems are backed by a powerful theory that can be used to study stability and reachability of mixed continuous and discrete systems. Thus, we can use this theory in cooperative robotics to detect deadlocks, test reachability of undesired or target states and study the stability of the cooperative system. For this, it should be necessary to abstract our representation trying to obtain simpler models (for example a linear automata) from our general hybrid automata, in order to use the tools already available for the analysis of these classes of hybrid systems. We also believe that with the advances in both hybrid systems theory and in computer technology, we may soon have software tools and computer power to formally analyze general hybrid systems models.

Another important future work is to experiment the role assignment mechanism in other cooperative tasks and with larger groups of real robots. The main difficulty of real implementations is to have large teams of real robots available. One solution is to build several Lego robots, similar to the ones presented in [Pereira et al., 2002], and use them in cooperative tasks. Lego robots are relatively cheap and easy to build, but they have limitations in sensor, actuation, and communication capabilities. Another possibility is continue this work in cooperation with other laboratories. For example, the GRASP Laboratory, where part of this work has been developed, has plans to build ten Clodbuster robots that would be a good testbed for experiments using the role assignment.

Still regarding the dynamic role assignment mechanism, an aspect that we have not addressed in this thesis and that should be investigated in future research is the choice of suitable utility functions for the role reallocation. In the cooperative search and transporta-

tion presented in Chapter 5, we used a simple heuristic function for the role reallocation. We believe that using some optimization technics applied to each task objective, it would be possible to implement tailored functions, thus obtaining a better performance for a given task.

In this work, we are making a strong assumption that sensors, plans and communication are almost perfect and errors do not influence the cooperative task execution. Consequently, the robots are able to know their exact location using odometry, can accurately detect obstacles using their cameras or sonar and can transmit their data without failures. In general, these assumptions are not realistic. Robots are inherently uncertain about their state and the state of the environment. Uncertainty in mobile robotics arises from different sources like sensors, actuators, and communication mechanisms. So, an important extension of this work is to consider these types of uncertainties, observing how the coordination mechanism would behave under these new conditions and trying to cope with them. In Appendix A we present some of the approaches that have been used in robotics to deal with these uncertainties and can be applied to our methodology in the future.

Another objective we would like to accomplish in a near future is to develop a "release version" of the simulator. In its current stage, due to several modifications implemented during the development of this work, the readability of MuRos code is somewhat compromised. Also, for now, to implement new applications using the simulator the user must have access to all the code to create new classes according to his/her necessity. In spite of the object orientation, sometimes users have to modify some of the base classes in order to implement their applications. We would like to avoid this, encapsulating the base classes in a class library that will be used to develop new applications under MuRoS. We also want to better organize the code, write a technical documentation (user manual) about the simulator and make it available on the World Wide Web. This will make MuRoS more usable and portable and will allow other research groups to use it and collaborate on its development.

Finally, a very interesting possibility of research is to increase the integration between

the hybrid systems modeling and the simulator. As shown in this thesis, simulations developed in MuRoS already benefit from the hybrid systems modeling: users can define roles, transitions and synchronize transitions using explicit communication. But everything should be explicitly implemented in the simulation code. A very useful feature is to automatically generate code for the simulator from high level specifications. In this way, the users would simply give a high level modeling of the cooperative task (for example the hybrid automaton of each robot) and the simulation code in MuRoS would be automatically generated. In fact, some tools like Charon [Alur et al., 2000a] and Stateflow [Mathworks, 2001] have already started addressing this problem, but none of them are specifically implemented for cooperative robotics. The "Holy Grail" of multi-robot programming (and mobile robotics programming in general) is to be able to, from high level specifications such as a hybrid automaton, automatically generate code to control different types of real robots acting in real environments and performing several types of tasks. This would certainly be a huge advance in the modeling and implementation of cooperative robotics.

Bibliography

[Alur et al., 1995] Alur, R., Coucoubetis, C., Henzinger, T. A., Ho, P.-H., Nicollin, X., Olivero, A., Sifakis, J., and Yovine, S. (1995). The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3–34.

[Alur et al., 2001] Alur, R., Das, A., Esposito, J., Fierro, R., Grudic, G., Hur, Y., Kumar, V., Lee, I., Ostrowski, J., Pappas, G., Southall, B., Spletzer, J., and Taylor, C. (2001). A framework and architecture for multirobot coordination. In Rus, D. and Singh, S., editors, *Experimental Robotics VII, LNCIS 271*. Springer Verlag.

[Alur and Dill, 1994] Alur, R. and Dill, D. (1994). A theory of timed automata. *Theoretical Computer Science*, 126:183–235.

[Alur et al., 2000a] Alur, R., Grosu, R., Hur, Y., Kumar, V., and Lee, I. (2000a). Modular specification of hybrid systems in charon. In *Proceedings of the 3rd International Workshop on Hybrid Systems: Computation and Control*.

[Alur et al., 2000b] Alur, R., Henzinger, T. A., Lafferriere, G., and Pappas, G. (2000b). Discrete abstractions of hybrid systems. *Proceedings of the IEEE*, 88(7):971–984.

[Antsaklis, 2000] Antsaklis, P. (2000). Special issue on hybrid systems: theory and applications a brief introduction to the theory and applications of hybrid systems. *Proceedings of the IEEE*, 88(7):879–887.

[Arkin et al., 1993] Arkin, R., Balch, T., and Nitz, E. (1993). Communication of behavioral state in multiagent retrieval tasks. In *Proceedings of the 1993 IEEE International Conference on Robotics and Automation*.

[Arkin, 1998] Arkin, R. C. (1998). *Behavior-Based Robotics*. MIT Press, Cambridge, MA.

[Asama et al., 1989] Asama, H., Matsumoto, A., and Ishida, Y. (1989). Design of an autonomous and distributed robot systems: Actress. In *Proceedings of the 1989 IEEE/RJS International Conference on Intelligent Robots and Systems*, pages 283–290.

[Babaoglu and Marzullo, 1993] Babaoglu, O. and Marzullo, K. (1993). Consistent global states of distributed systems: Fundamental concepts and mechanisms. In Mullender, S., editor, *Distributed Systems*, pages 55–96. Addison-Wesley.

[Balch and Arkin, 1995] Balch, T. and Arkin, R. C. (1995). Communication in reactive multiagent robotic systems. *Autonomous Robots*, 1(1):27–52.

[Balch and Arkin, 1998] Balch, T. and Arkin, R. C. (1998). Behavior-based formation control for multirobot teams. *IEEE Transactions on Robotics and Automation*, 14(6).

[Balch and Hybinette, 2000] Balch, T. and Hybinette, M. (2000). Social potentials for scalable multirobot formations. In *Proceedings of the 2001 IEEE International Conference on Robotics and Automation*, pages 73–80.

[Borenstein and Feng, 1996] Borenstein, J. and Feng, L. (1996). Measurement and correction of systematic errors in mobile robots. *IEEE Transactions on Robotics and Automation*, 12(6):869–880.

[Brooks, 1986] Brooks, R. (1986). A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2(1):14–23.

[Brooks, 1991] Brooks, R. (1991). Intelligence without reason. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, pages 569–595.

[Brown and Jennings, 1995] Brown, R. G. and Jennings, J. S. (1995). A pusher/steerer model for strongly cooperative mobile robot manipulation. In *Proceedings of the 1995 IEEE/RJS International Conference on Intelligent Robots and Systems*, volume 3, pages 562–568.

[Burgard et al., 1996] Burgard, W., Fox, D., Hennig, D., and Schmidt, T. (1996). Estimating the absolute position of a mobile robot using position probability grids. In *Proceedings of the 1996 National Conference on Artificial Intelligence.*

[Burgard et al., 2001] Burgard, W., Fox, D., and S., T. (2001). Probabilistic techniques for mobile robots. Tutorial Notes - 2001 International Conference on Robotics and Automation.

[Cao et al., 1997] Cao, Y. U., Fukunaga, A. S., and Kahng, A. B. (1997). Cooperative mobile robotics: Antecedents and directions. *Autonomous Robots*, 4:1–23.

[Castellanos et al., 1999] Castellanos, J., Montiel, J., Neira, J., and Tardos, J. (1999). The spmap: A probabilistic framework for simultaneous localization and mapping building. *IEEE Transactions on Robotics and Automation*, 15(5):948–953.

[Castelpietra et al., 2000] Castelpietra, C., Iocchi, L., Nardi, D., and Rosati, R. (2000). Coordination in multi-agent autonomous cognitive robotic systems. In *Proceedings of 2nd International Cognitive Robotics Workshop.*

[Chaimowicz et al., 2001a] Chaimowicz, L., Campos, M., and Kumar, V. (2001a). Simulating loosely and tightly coupled multi-robot cooperation. In *Proceedings of V SBAI – Brazilian Symposium on Intelligent Automation.*

[Chaimowicz et al., 2002] Chaimowicz, L., Campos, M., and Kumar, V. (2002). Dynamic role assignment for cooperative robots. In *Proceedings of the 2002 IEEE International Conference on Robotics and Automation*, pages 292–298.

[Chaimowicz et al., 2001b] Chaimowicz, L., Kumar, V., and Campos, M. (2001b). A framework for coordinating multiple robots in cooperative manipulation tasks. In *Proceedings of SPIE Vol. 4571 – Sensor Fusion and Decentralized Control IV*, pages 120–127.

[Chaimowicz et al., 2001c] Chaimowicz, L., Sugar, T., Kumar, V., and Campos, M. (2001c). An architecture for tightly coupled multi-robot cooperation. In *Proceedings of the 2001 IEEE International Conference on Robotics and Automation*, pages 2292–2297.

[Cox, 1991] Cox, I. J. (1991). Blanche - an experiment in guidance and navigation of an autonomous robot vehicle. *IEEE Transactions on Robotics and Automation*, 7(2):193–204.

[Das et al., 2001] Das, A., Fierro, R., Kumar, V., Southall, B., Spletzer, J., and Taylor, C. (2001). Real-time vision-based control of a nonholonomic mobile robot. In *Proceedings of the 2001 IEEE International Conference on Robotics and Automation*, pages 1714–1719.

[De Luca and Oriolo, 1994] De Luca, A. and Oriolo, G. (1994). Local incremental planning for nonholonomic mobile robots. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*.

[de Weerdt, 1998] de Weerdt, M. M. (1998). Specifying uncertainty for mobile robots. Master's thesis, Utrecht University.

[Desai et al., 1999] Desai, J. P., Kumar, V., and Ostrowski, J. (1999). Control of changes in formation for a team of mobile robots. In *Proceedings of the 1999 IEEE International Conference on Robotics and Automation*, pages 1556–1561.

[Donald, 1995] Donald, B. R. (1995). Information invariants in robotics. *Artificial Intelligence Journal*, 72:217–304.

[Donald et al., 2000] Donald, B. R., Gariepy, L., and Rus, D. (2000). Distributed manipulation of multiple objects using ropes. In *Proceedings of the 2000 IEEE International Conference on Robotics and Automation*, pages 450–456.

[Donald et al., 1997] Donald, B. R., Jennings, J., and Rus, D. (1997). Information invariants for distributed manipulation. *The International Journal of Robotics Research*, 16(5):673–702.

[Drogoul and Ferber, 1992] Drogoul, A. and Ferber, J. (1992). From tom thumb to the dockers: Some experiments with foraging robots. In *Proceedings of the Second International Conference on Simulation of Adaptive Behavior*, pages 451–459.

[Dudek and Jenkin, 2000] Dudek, G. and Jenkin, M. (2000). *Computational Principles of Mobile Robots*. Cambridge University Press.

[Dudek et al., 1996] Dudek, G., Jenkin, M., Milios, E., and Wilkes, D. (1996). A taxonomy for multi-agent robotics. *Autonomous Robots*, 3(4):375–397.

[Egerstedt and Hu, 2002] Egerstedt, M. and Hu, X. (2002). A hybrid control approach to action coordination for mobile robots. *Automatica*, 38(1):125–130.

[Elfes, 1989] Elfes, A. (1989). *Occupancy Grids: A Probabilistic Framework for Robot Perception and Navigation*. PhD thesis, Department of Electrical and Computer Engineering - Carnegie Mellon University.

[Emery et al., 2002] Emery, R., Sikorski, K., and Balch, T. (2002). Protocols for collaboration, coordination and dynamic role assignment in a robot team. In *Proceedings of the 2002 IEEE International Conference on Robotics and Automation*, pages 3008–3015.

[Espiau et al., 1995] Espiau, B., Kapellos, K., Jourdan, M., and Simon, D. (1995). On the validation of robotics control systems. part i: High level specification and formal verification. Technical Report 2719, INRIA - Rhône-Alpes.

[Fox et al., 1999] Fox, D., Burgard, W., Dellaert, F., and S., T. (1999). Monte carlo localization: Efficient position estimation for mobile robots. In *Proceedings of the 1999 National Conference on Artificial Intelligence (AAAI)*.

[Fukuda and Nakagawa, 1987] Fukuda, T. and Nakagawa, S. (1987). A dynamically reconfigurable robotic system (concept of a system and optimal configurations). In *Proceedings of IECON Robotics and Automation*, pages 588–595.

[Garg, 1996] Garg, V. (1996). *Principles of Distributed Systems*. Kluwer Academic Publishers, Norwell, Massachusetts.

[Gerkey and Mataric, 2001] Gerkey, B. and Mataric, M. (2001). Principled communication for dynamic multi-robot task allocation. In Rus, D. and Singh, S., editors, *Experimental Robotics VII, LNCIS 271*, pages 353–362. Springer Verlag.

[Gerkey and Mataric, 2002] Gerkey, B. and Mataric, M. (2002). Pusher-watcher: An approach to fault-tolerant tightly-coupled robot coordination. In *Proceedings of the 2002 IEEE International Conference on Robotics and Automation*, pages 464–469.

[Gerkey et al., 2001] Gerkey, B., Vaughan, R., Stoy, K., Howard, A., Sukhatme, G., and Mataric, M. (2001). Most valuable player: A robot device server for distributed control. In *Proceedings of the Second International Workshop on Infrastructure for Agents, MAS, and Scalable MAS at Autonomous Agents 2001*.

[Gutmann et al., 1998] Gutmann, J., Burgard, W., Fox, D., and Konolige., K. (1998). An experimental comparison of localization methods. In *Proceedings of the 1998 IEEE/RSJ International Conference on Intelligent Robots and Systems*.

[Henzinger, 1996] Henzinger, T. (1996). The theory of hybrid automata. In *Proceedings of the 11th Annual Symposium on Logic in Computer Science*, pages 278–292.

[Henzinger et al., 1997] Henzinger, T., Ho, P., and Wong-Toi, H. (1997). Hytech: a model checker for hybrid systems. *Software Tools for Technology Transfer*, 1:110–122.

[Hirata and Kosuge, 2000] Hirata, Y. and Kosuge, K. (2000). Distributed robot helpers handling a single object in cooperation with a human. In *Proceedings of the 2000 IEEE International Conference on Robotics and Automation*, pages 458–463.

[Jennings and Kirkwood-Watts, 1998] Jennings, J. and Kirkwood-Watts, C. (1998). Distributed mobile robots by the method of dynamic teams. In *Distributed Autonomous Robotic Systems 3*. Springer Verlag.

[Jennings et al., 1997] Jennings, J., Whelan, G., and Evans, W. (1997). Cooperative search and rescue with a team of mobile robots. In *Proceedings of the IEEE International Conference on Advanced Robotics*, pages 193–200.

[Jensfelt and Kristensen, 1999] Jensfelt, P. and Kristensen, S. (1999). Active global localization for a mobile robot using multiple hypothesis tracking. In *Proceedings of the International Joint Conference on Artificial Intelligence*.

[Kalman, 1960] Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. *Transactions of ASME - Journal of Basic Engineering*, 82:35–45.

[Katz and Gil, 1999] Katz, S. and Gil, J. (1999). Aspects and superimpositions. In *ECOOP Workshops*, pages 308–309.

[Khatib, 1986] Khatib, O. (1986). Real-time obstacle avoidance for manipulators and mobile robots. *The International Journal of Robotics Research*, 05(1):90–99.

[Khatib et al., 1995] Khatib, O., Yokoi, K., Chang, K., Ruspini, D., Holmberg, R., Casal, A., and Baader, A. (1995). Force strategies for cooperative tasks in multiple mobile manipulation systems. In *Proceedings of the 7th International Symposium on Robotics Research*, pages 333–342.

[Kosuge and Oosumi, 1996] Kosuge, K. and Oosumi, T. (1996). Decentralized control of multiple robots handling an object. In *Proceedings of the 1996 IEEE/RJS International Conference on Intelligent Robots and Systems*, pages 318–323.

[Kosuge et al., 1998] Kosuge, K., Oosumi, T., Hirata, Y., Asama, H., Kaetsu, H., and Kawabata, K. (1998). Handling a single object by multiple autonomous mobile robots in coordination with body force sensor. In *Proceedings of the 1998 IEEE/RJS International Conference on Intelligent Robots and Systems*, pages 1419–1424.

[Kruglinski et al., 1998] Kruglinski, D., Wingo, S., and Sheperd, G. (1998). *Programming Visual C++*. Microsoft Press, 5 edition.

[Kube and Zhang, 1996] Kube, R. C. and Zhang, H. (1996). The use of perceptual cues in multi-robot box-pushing. In *Proceedings of the 1996 IEEE International Conference on Robotics and Automation*, pages 2085–2090.

[Kube and Zhang, 1997] Kube, R. C. and Zhang, H. (1997). Task modeling in collective robotics. *Autonomous Robots*, 4:53–72.

[Kume et al., 2001] Kume, Y., Hirata, Y., Kosuge, K., Asama, H., Kaetsu, H., and Kawabata, K. (2001). Decentralized control of multiple mobile robots transporting a single object in coordination without using force/torque sensors. In *Proceedings of the 2001 IEEE International Conference on Robotics and Automation*, pages 3004–3008.

[Leonard et al., 1992] Leonard, J., Durrant-Whyte, H., and Cox, I. J. (1992). Dynamic map building for an autonomous mobile robot. *International Journal of Robotics Reasearch*, 11(4):89–96.

[Lin and Hsu, 1995] Lin, F.-C. and Hsu, J. Y.-J. (1995). Cooperation and deadlock-handling for an object-sorting task in a multi-agent robotic system. In *Proceedings of the 1995 IEEE International Conference on Robotics and Automation*, pages 2580–2585.

[Mataric, 1994] Mataric, M. (1994). *Interaction and Intelligent Behavior*. PhD thesis, MIT.

[Mataric et al., 1995] Mataric, M. J., Nilsson, M., and Simsarian, K. T. (1995). Cooperative multi-robot box-pushing. In *Proceedings of the 1995 IEEE/RJS International Conference on Intelligent Robots and Systems*, volume 3, pages 556–561.

[Mathworks, 2001] Mathworks (2001). http://www.mathworks.com.

[McKerrow, 1991] McKerrow, P. (1991). *Introduction to Robotics*. Addison-Wesley, Reading, MA.

[Milutinovic and Lima, 2002] Milutinovic, D. and Lima, P. (2002). Petri net models of robotic tasks. In *Proceedings of the 2002 IEEE International Conference on Robotics and Automation*, pages 4059–4064.

[Mullender, 1993] Mullender, S. (1993). *Distributed Systems*. ACM Press, NYC, 2nd edition.

[Nomadic, 1993] Nomadic (1993). *Nomad 200 User's Guide*. Nomadic Technologies, Mountain View, CA.

[Noreils, 1993] Noreils, F. R. (1993). Toward a robot architecture integrating cooperation between mobile robots: Application to indoor environment. *The International Journal of Robotics Research*, 12(1):79–98.

[Ollennu et al., 2002] Ollennu, A., Nayar, H., Aghazarian, H., Ganino, A., Pirjanian, P., Kennedy, B., Huntsberger, T., and Schenker, P. (2002). Mars rover pair cooperatively transporting a long payload. In *Proceedings of the 2002 IEEE International Conference on Robotics and Automation*, pages 3136–3141.

[Ota et al., 1995] Ota, J., Miyata, N., Arai, T., Yoshida, E., Kurabayashi, D., and Sasaki, J. (1995). Transferring and regrasping a large object by cooperation of multiple mobile robots. In *Proceedings of the 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 543–548.

[Parker, 1994] Parker, L. E. (1994). *Heterogeneous Multi-Robot Cooperation*. PhD thesis, Massachusetts Institute of Technology.

[Parker, 1995] Parker, L. E. (1995). The effect of action recognition and robot awareness in cooperative robotic teams. In *Proceedings of the 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 212–219.

[Parker, 1998] Parker, L. E. (1998). Alliance: An architecture for fault tolerant multi-robot cooperation. *IEEE Transactions on Robotics and Automation*, 14(2):220–240.

[Parker, 2000] Parker, L. E. (2000). Current state of the art in distributed robot systems. In Parker, L. E., Bekey, G., and Barhen, J., editors, *Distributed Autonomous Robotic Systems 4*, pages 3–12. Springer Verlag.

[Pereira et al., 2002] Pereira, G., Pimentel, B., Chaimowicz, L., and Campos, M. (2002). Coordination of multiple mobile robots in an object carrying task using implicit communication. In *Proceedings of the 2002 IEEE International Conference on Robotics and Automation*, pages 281–286.

[Pimentel, 2002] Pimentel, B. (2002). Cooperation for communication: Exploring ad hoc connectivity in a group of cooperating mobile robots. Master's thesis, Compute Science Department - Federal University of Minas Gerais (DCC - UFMG) - Brazil.

[Press et al., 1988] Press, H., Flannery, B., Teukolsky, S., and Vetterling, W. (1988). *Numerical Recipes in C*. Cambridge University Press, New York, NY.

[Rus et al., 1995] Rus, D., Donald, B., and Jennings, J. S. (1995). Moving furniture with teams of autonomous robots. In *Proceedings of the 1995 IEEE/RJS International Conference on Intelligent Robots and Systems*, volume 1, pages 235–243.

[Shehory and Kraus, 1998] Shehory, O. and Kraus, S. (1998). Methods for task allocation via agent coalition formation. *Artificial Intelligence Journal*, 101(1-2):165–200.

[Simmons and Koenig, 1995] Simmons, R. and Koenig, S. (1995). Probabilistic robot navigation in partially observable environments. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1080–1087.

[Song et al., 2001] Song, P., Kraus, P., Kumar, V., and Dupont, P. (2001). Analysis of rigid body dynamic models for simulation of systems with frictional contacts. *ASME Journal of Applied Mechanics*, 68(1):118–128.

[Song and Kumar, 2002] Song, P. and Kumar, V. (2002). A potential field based approach to multi-robot manipulation. In *Proceedings of the 2002 IEEE International Conference on Robotics and Automation*, pages 1217–1222.

[Stilwell and Bay, 1993] Stilwell, D. J. and Bay, J. S. (1993). Towards the development of a material transport system using swarms of ant-like robots. In *Proceedings of the 1993 IEEE International Conference on Robotics and Automation*, pages 766–771.

[Stone and Veloso, 1999] Stone, P. and Veloso, M. (1999). Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork. *Artificial Intelligence*, 110(2):241–273.

[Sugar, 1999] Sugar, T. (1999). *Design and Control of Cooperative Mobile Robotic Systems*. PhD thesis, University of Pennsylvania.

[Sugar and Kumar, 1998a] Sugar, T. and Kumar, V. (1998a). Decentralized control of cooperating mobile manipulators. In *Proceedings of the 1998 IEEE International Conference on Robotics and Automation*, pages 2916–2921.

[Sugar and Kumar, 1998b] Sugar, T. and Kumar, V. (1998b). Design and control of a compliant parallel manipulator for a mobile platform. In *Proceedings of the 1998 ASME Design Engineering Technical Conferences and Computers in Engineering Conference*.

[Sugar and Kumar, 1999] Sugar, T. and Kumar, V. (1999). Multiple cooperating mobile manipulators. In *Proceedings of the 1999 IEEE International Conference on Robotics and Automation*, pages 1538–1543.

[Tambe, 1997] Tambe, M. (1997). Agent architecture for flexible, practical teamwork. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*.

[TeamBots, 2001] TeamBots (2001). http://www.teambots.org.

[Thrun, 2000] Thrun, S. (2000). Probabilistic algorithms in robotics. *AI Magazine*, 21(4):93–109.

[Thrun et al., 1998] Thrun, S., Fox, D., and Burgard, W. (1998). A probabilistic approach to concurrent mapping and localization for mobile robots. *Machine Learning*, 31:29–53.

[Uchibe et al., 2001] Uchibe, E., Kato, T., Asada, M., and Hosada, K. (2001). Dynamic task assignment in a multiagent/multitask environment based on module conflict resolution. In *Proceedings of the 2001 IEEE International Conference on Robotics and Automation*, pages 3987–3992.

[Wang and Saridis, 1993] Wang, F. and Saridis, G. (1993). Task translation in integration specification in intelligent machines. *IEEE Transactions on Robotics and Automation*, 9(3):257–271.

[Wang et al., 1996] Wang, Z., Nakano, E., and Matsukawa, T. (1996). Realizing cooperative object manipulation using multiple behavior-based robots. In *Proceedings of the 1996 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 310–317.

[Werger and Mataric, 2000] Werger, B. B. and Mataric, M. J. (2000). Broadcast of local eligibility for multi-target observation. In Parker, L. E., Bekey, G., and Barhen, J., editors, *Distributed Autonomous Robotic Systems 4*. Springer Verlag.

[Ye et al., 2001] Ye, W., Vaughan, R., Sukhatme, G., Heidemann, J., Estrin, D., and Mataric, M. (2001). Evaluating control strategies for wireless-networked robots using an integrated robot and network simulator. In *Proceedings of the 2001 IEEE International Conference on Robotics and Automation*, pages 2941–2947.

Appendix A

Dealing with Uncertainty

In our coordination mechanism, robots depend on local and global information in order to make decisions regarding the role assignment and other aspects of the task execution. Normally, the robots use several kinds of sensors (cameras, sonar, laser, odometry) in order to gather information, rely on pre-computed maps and plans, and use explicit communication mechanisms to exchange information with other robots. In this thesis, we made a strong assumption that sensors, plans, and communication are almost perfect and errors do not influence the cooperative task execution. In general situations, this assumption is not realistic and robots are inherently uncertain about their state and the state of the environment. So, in spite of not considering these factors in our coordination mechanism so far, we think that it is important to present some of the approaches that have been used in robotics to deal with these uncertainties.

Uncertainty arises from different sources. First of all, robots depend on their sensors to gather information about their state and the environment. In a very broad sense, we can say that sensors are noisy, return incomplete descriptions of the environment and cannot usually be modeled completely [Dudek and Jenkin, 2000]. In general, a single sensor measurement is not sufficient to correctly estimate the robot's state and gather reliable information from the environment. Each sensor category has its limitations: odometry, for example, is subjected to systematic errors (such as misalignment of robot wheels) and

non-systematic errors (such as wheel slippage) [Borenstein and Feng, 1996]. Sonar sensors have a limited range and can be influenced by specular reflections, surface properties, etc. Misinterpretation of sensor data can also lead to incorrect beliefs. Other important sources of uncertainty also exist. In general, the use of a priori and static models brings uncertainty to the system because prior knowledge about the environment may be incorrect and real world environments usually have unpredictable dynamics. For example, maps omit details, metric information can be imprecise, objects move, and even relatively stable features may change over time [de Weerdt, 1998]. Also, the effects of control actions are not completely reliable: wheels slip, controllers are affected by noise, etc. Summarizing, we can say that both the robots and the environment are stochastic, sensors are limited and noisy, models are inaccurate, and all these factors lead to a high level of uncertainty in mobile robotics.

A large group of researchers have been studying how to deal with uncertainty, making this a very broad and active research area. Most of the material discussed in this section was compiled from a recent survey [Thrun, 2000] and from a tutorial presented during the 2001 International Conference of Robotics and Automation [Burgard et al., 2001]. Other references are also cited throughout the text.

The dominant approach that has been used to deal with uncertainty is called Probabilistic Robotics. The main idea is to explicitly represent uncertainty through the probability theory, using probability densities to represent information. The central conjecture of this approach can be stated as follows: A robot that carries a notion of its own uncertainty and that acts accordingly will do better that one that does not. The main advantages of probabilistic approaches are that they can accommodate inaccurate models and imperfect sensors acting robustly in real world applications. On the other hand, probabilistic algorithms tend to be computationally demanding, need to perform approximations in order to represent the world and depend on some assumptions that may not be completely true in all situations.

One of the areas where probabilistic robotics has been used with great success is in the localization problem. In this problem, robots must find their coordinates relative to the environment assuming that they have a map of this environment. The localization

problem can be categorized according to different difficulty levels. The easier situation is when the robot knows precisely its initial location, and needs only to maintain an accurate estimate of its current pose. This problem is known as position tracking. When the robot does not have information about its initial location, it faces a more difficult localization problem, in which it must find its pose from scratch, relying only on its sensors. This is called global localization. It is important to mention that some researchers consider the localization problem the most fundamental problem that needs to be solved to provide a mobile robot with autonomous capabilities [Cox, 1991].

A successful family of probabilistic approaches that has been developed and used by several researchers to solve both localization problems is called Markov localization. The central idea of Markov localization is to represent the robot's belief by a probability distribution over possible positions and to use Bayes' rule and convolution to update the belief whenever the robot senses or moves [Fox et al., 1999]. The general equation for the Markov localization comes from Bayes' filters. Considering that x denotes a position in the state space of the robot, m is the map of the environment and a_i and s_i $(i = 0, \ldots, t)$, represent respectively consecutive robot actions and observations, the distribution $bel(x_t|m)$ expresses the robot's belief for being at position x at time t given the map m, and can be defined as:

$$bel(x_t|m) = p(x_t|s_0, \ldots, a_{t-1}, s_t, m)$$

To estimate this equation, probabilistic approaches normally resort to a Markov assumption, which states that the future is independent of the past given knowledge of the current state. Hence, it is assumed that the observations are independent (and so the noise), and that the robot's pose is the only state that would impact more than just one isolated sensor reading. This is also known as the static world assumption. Based on the Markov assumption, and applying Bayes' rule and the total probability theorem, the belief

$bel(x_t|m)$ can be computed using a recursive formula as shown below:

$$bel(x_t|m) = \eta p(s_t|s_0, \ldots, a_{t-1}, x_t, m) p(x_t|s_0, \ldots, a_{t-1}, m) \quad \text{(Bayes)}$$

$$bel(x_t|m) = \eta p(s_t|x_t, m) p(x_t|s_0, \ldots, a_{t-1}, m) \quad \text{(Markov)}$$

$$bel(x_t|m) = \eta p(s_t|x_t, m) \int p(x_t|s_0, \ldots, a_{t-1}, x_{t-1}, m) p(x_{t-1}|s_0, \ldots, s_{t-1}, a_{t-1}, m) dx_{t-1} \quad \text{(T. Prob.)}$$

$$bel(x_t|m) = \eta p(s_t|x_t, m) \int p(x_t|a_{t-1}, x_{t-1}, m) p(x_{t-1}|s_0, \ldots, a_{t-2}, s_{t-1}, m) dx_{t-1} \quad \text{(Markov)}$$

$$bel(x_t|m) = \eta p(s_t|x_t, m) \int p(x_t|a_{t-1}, x_{t-1}, m) bel(x_{t-1}|m) dx_{t-1}$$

The constant η is a normalizer that ensures that the result sums up to one. To implement this last equation, two probabilistic densities must be specified: the first one, $p(s_t|x_t, m)$, is a probabilistic model of the perception and the second one, $p(x_t|a_{t-1}, x_{t-1}, m)$, characterizes the effect of robot actions on its pose, and can be considered as a probabilistic generalization of the robot's kinematics. Both densities are usually time-invariant, so the time index t can be omitted.

An important point in the Markov localization is the definition of a suitable representation for the belief. One of the more traditional approaches is to use Kalman filters [Kalman, 1960], assuming that the uncertainty in the robot's position can be described by a unimodal Gaussian distribution. This kind of approach has been used successfully in various robot applications for position tracking, for example in [Gutmann et al., 1998]. This kind of representation, however, has some drawbacks. In practice, localization approaches using Kalman filters require that the starting position of the robot is known, making difficult its use on the global localization problem. In addition, Kalman filters rely on sensor models that generate estimates with Gaussian uncertainty, which is often unrealistic [Fox et al., 1999]. Other possible representations include multi-hypothesis tracking [Jensfelt and Kristensen, 1999], topological representations [Simmons and Koenig, 1995], and grid-based representations [Burgard et al., 1996]. A very powerful approach for the representation is to approximate the belief using a weighted set of samples so that the discrete distribution defined by the samples approximates the desired one. This technique

is generically known as importance sampling, but is also known with other names according to the area of application (for example, particle filters or condensation algorithm). In robotics, it is sometimes called Monte Carlo localization and has some advantages over other representations [Fox et al., 1999]: it is easier to implement, allows the representation of multimodal distributions, can globally localize a robot and has improved efficiency in terms of memory and speed when compared to grid-based approaches.

Another area where probabilistic approaches have been used successfully is mapping. The mapping problem consists in creating a representation of the environment (map) based on information gathered with the robot sensors. Besides the difficulties regarding sensor noise and modeling, mapping can be a very hard task if the robot does not have precise information about its current pose. As mentioned in [Thrun, 2000], this problem has a "chicken and egg" nature: localization with a map is relatively easy, as is mapping with known locations. Combined, however, they turn into a difficult problem.

In combination, however, this problem is hard.

One of the seminal works in probabilistic mapping is the occupancy grid approach [Elfes, 1989]. Basically, this approach decomposes the world in a grid of cells and the value of each cell represents the probability of it being occupied or not. The occupancy values are updated with consecutive sensor readings using an equation based on Bayes' rule. Occupancy grids is one of the most used mapping algorithms when a good estimation of the robot's pose is available. Normally it is augmented with ad-hoc methods for localization during mapping. Other common algorithm uses a similar approach for mapping as for localization. The basic idea is to include the map together with the robot's state in the computation of the belief. Examples of this approach using Kalman filters to represent the belief can be found in [Castellanos et al., 1999] and [Leonard et al., 1992]. These techniques are generically known as SLAM (simultaneous localization and mapping). Extensions of these algorithms using other representations for belief have also been developed, for example, using the EM (expectation-maximization) algorithm [Thrun et al., 1998].

Appendix B

MuRoS - A Multi-Robot Simulator

All the simulations presented in this thesis were performed using MuRoS, the simulator the we have developed for cooperative robotics. In this appendix, we describe the simulator in details. We begin by giving a general description of the simulator and discussing other simulators available for cooperative robotics. Then we explain MuRoS structure in terms of its hierarchy, control, communication and sensing mechanisms, and finish showing some examples of its use in the simulation of cooperative tasks.

B.1 General Description

MuRoS (Multi-Robot Simulator) is a software tool that we have developed for the simulation of cooperative robotics. It allows the simulation of various types of tasks, ranging from loosely coupled to tightly coupled cooperative tasks. Developed using object oriented programming in Visual C++ for the MS Windows environment, MuRoS allows the simulation of several multi-robot applications such as cooperative manipulation, formation control, foraging, etc. It allows the creation of different types of robots and obstacles and the observation of the simulation in real time. Also, result data can be exported to other tools such as Matlab for future analysis. New types of robots can be implemented with different controllers, driving mechanisms and sensors, inheriting characteristics from other

robots. Implicit and explicit communication can be simulated, allowing the robots to exchange information during the task execution. All these characteristics make MuRoS a suitable tool for simulation of cooperative robotic systems.

B.2 Simulators for Cooperative Robotics

Some general simulators for cooperative robotics can be found in the literature and are available for use. One example is the Teambots platform [TeamBots, 2001], in which teams of robots controlled by behavior-based approaches can be simulated. This platform is implemented in Java and contains several classes that allows the implementation of different multi-robot scenarios. One example of these scenarios can be found in [Balch and Hybinette, 2000]. Other software is Stage, which simulates mobile robots sensing and moving in a two-dimensional bitmapped environment, controlled through a device server called Player [Gerkey et al., 2001]. An extension that groups a previous version of Stage with a wireless network simulator is described in [Ye et al., 2001] and is used in the study of communication protocols for cooperative robotics. Another interesting approach is Charon, a tool for modeling and analyzing hybrid systems in general that can be used to simulate multi-robot coordination and control [Alur et al., 2000a]. Nomadic Technologies developed a simulator for its Nomad 200 robot [Nomadic, 1993]. This simulator is interesting because the control program being simulated can be used in the real robot without any modification, by simply linking the program with a different library. Several robots can be created in this simulator, allowing it to be used for multi-robot cooperation. Another simulation tool is MissionLab [Arkin, 1998], which is used for designing multi-agent missions. It allows the recursive definition of robot societies and includes a graphical configuration editor, a multi-agent simulation system, and code generators for two different architectures. Some results of its use in a formation control task can be found in [Balch and Arkin, 1998]. Finally, general software tools such as Matlab and Simulink [Mathworks, 2001] have also been used to simulate some specific cooperative robotic applications. Examples can be found in [Pimentel, 2002] and [Song and Kumar, 2002].

We have tried some of these simulators but none of them completely fulfilled our needs for simulations of the role assignment mechanism in teams of potentially heterogeneous robots. Some simulators are specific to a certain type of robot or controller, restricting their use. Others are too general, making the implementation of cooperative robotic tasks a difficult job. Furthermore, sometimes it is easier to start the development of a simulator from scratch than to use a simulator developed by other groups, due to the lack of documentation and problems in extending and reusing the source code. Consequently, we have preferred to start the development of a new simulator for cooperative robotics and to perform our experiments using it.

B.3 MuRoS Structure

As mentioned, MuRoS has been implemented using Visual C++. Visual C++ has a internal structure and an extensive API (Application Programming Interface) that facilitates the development of applications for the Windows environment. It is based on the concept of documents and views. Basically, the document class stores the application data and the view class is responsible for displaying it. Different view classes can be implemented for the same document, and MDI (Multiple Document Interface) applications can handle several documents simultaneously. An example of this structure is a text editor (such as MS Word), in which several files can be opened and edited simultaneously. More details about the document/view architecture can be found in [Kruglinski et al., 1998].

MuRoS uses the document/view structure, but only a single document is active during execution. Therefore, it is a SDI (Single Document Interface) application. The document class contains the main data structures of the simulator, namely: `m_robots`, `m_obstacles`, and `m_boxes`. These variables are arrays that store respectively the robots being simulated, the obstacles in the environment and possible boxes[1] to be transported in cooperative manipulation tasks. Using object orientation (inheritance and polymorphism) it is possible

[1] In this appendix, we are using the term "box" instead of "object" to avoid confusion with the term "object" in object oriented programming.

to store different types of robots, obstacles, and boxes in these structures.

The user interacts with these data structures using the graphical interface. Selecting the appropriate buttons, it is possible to create robots, obstacles, and boxes by clicking at the desired positions in the application screen. After creating the simulation elements (robots, obstacles, and boxes), the user can start the simulation by choosing the start button on the interface. At this point, a new thread is created in order to integrate the differential equations that control each robot. This thread is basically a loop that increments the simulation time by a step factor and calls the `Update` method of each element in the `m_robots` and `m_boxes` arrays. This method is responsible for computing the velocities and positions of the robots and boxes integrating the dynamic equations defined for them. In fact, the `Update` method is a virtual function that is redefined in each class according to the desired dynamics of the robots and boxes. For now, we are using the Euler method [Press et al., 1988] for integration. Despite its simplicity, this method has been adequate for integrating the differential equations used thus far and has good performance when compared to higher order integration methods. In each simulation step, after calling the `Update` methods, the thread calls another method to redraw the simulation screen. This method calls the `Draw` method of each object, another virtual function implemented in each class, that is responsible for drawing the elements on the screen. If necessary, the rate in which the screen is redrawn can be reduced, improving the performance of the simulator.

The use of two different threads, one for running the simulation and other for the user interface, leads to an excellent performance and allows the access to the data structures during the simulation. For example, the user can add or remove robots and obstacles, and change control parameters while the simulation is running.

B.4 Class Hierarchy

One of the fundamental concepts in object oriented programming is inheritance. The inheritance mechanism allows the developer to create base classes with general characteristics of an entity and specialize them in new subclasses, with the addition of attributes, methods

and the redefinition of virtual functions. It is a very important concept, mainly when code reuse and extensibility are necessary.

In MuRoS, we have implemented a class hierarchy that allows the creation of new types of robots inheriting characteristics from classes that have already been developed. The main class is `CRobot` that contains the basic attributes, methods, and virtual functions of a generic robot. The attributes include, for example, the position of the robot, the number of obstacles detected by its sensors, etc. As mentioned in the previous section, two main virtual functions must be redefined: `Update` and `Draw`. Thus, to create a new type of robot, the user must create a new class that inherits from the `CRobot` class or from one of its subclasses, redefine some of its functions, and add methods and data members for new capabilities such as new sensors, actuators, etc.

Figure B.1 shows MuRoS's class hierarchy diagram. In the first level of inheritance, we have the subclasses `CRobotHolonomic` and `CRobotNonHolonomic`, with the specific characteristics of holonomic and non-holonomic robots respectively. From them, we can derive more specific classes, for example: `CRobotXR4000` representing the Nomadic Technologies' holonomic robot or `CRobotLabmate` representing a TRC Labmate platform equipped with a compliant arm [Sugar and Kumar, 1998b]. The class hierarchy also includes different types of obstacles, boxes, and a class for mapping and planning, that is used by some of the robots. Some internal classes of the simulator (for example the document and view classes explained before) are not shown in this diagram.

This class hierarchy is being continuously upgraded with the addition of new classes, methods, and attributes. As discussed in Chapter 7, one of our future directions is to create a fixed class library that will be used as base for the development of new classes and applications under MuRoS, to allow other research groups to use the simulator.

B.5 Controllers

The simulator allows the implementation of different types of robot controllers. We are using the distributed hybrid approach described in Chapter 4: the behavior of each robot

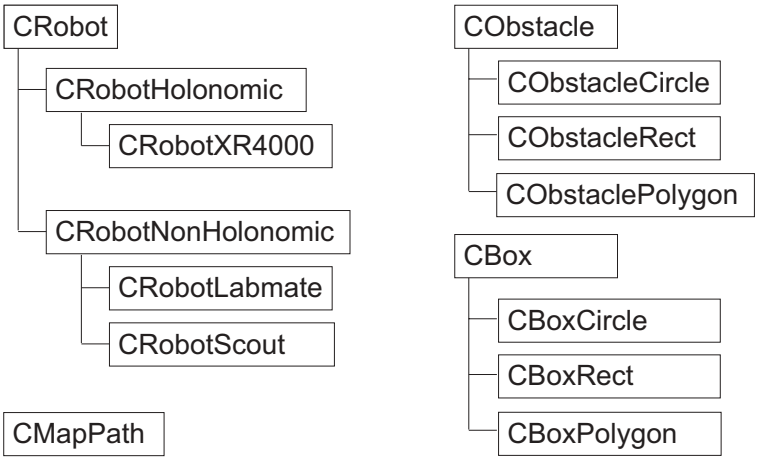


Figure B.1: Class hierarchy of the simulator.

is described by a hybrid automaton and robots can switch among different discrete states (modes), having different continuous controllers in each mode. Basically, the user defines in the `Update` method the differential equations that control the robots in each mode, and the simulator integrates these equations displaying the robots trajectories in real time. Switches between modes are triggered based on the robots continuous states, perception of the environment, and explicit communication. Several controllers have been experimented in the simulator, such as potential fields, path following, leader-follower, and open loop approaches.

We are using simple kinematic models to describe the robots in the simulator. Each robot is described by its pose $[x, y, \theta]^T$ and velocities $[v, \omega]^T$ for nonholonomic robots and $[v_x, v_y, \omega]^T$ for the holonomic ones. As explained in the previous sections, the poses are updated integrating the velocities using the Euler method. The velocities can either be set directly or be obtained from forces applied on the robots. In this case, we have a simple dynamic model where robots are subject to forces and torques and the velocities are computed integrating these forces. This is the case, for example, of the potential field method [Khatib, 1986] used by the robots to navigate through the environment in some applications. In this method, each robot is subject to attractive forces from the targets (for

example the items to be retrieved, the goal, etc.) and local repulsive forces from the other robots and obstacles. Forces due to physical contacts with obstacles and other robots are also computed using a simplified rigid body dynamic model based on [Song et al., 2001]. Basically, each robot computes the resultant of all these forces and integrates it to obtain the velocities. For the holonomic robots, the forces and torques $[F_x, F_y, \tau]^T$ are directly integrated giving $[v_x, v_y, \omega]^T$. For the non-holonomic robots, we implemented some controllers that are similar to the ones described in [De Luca and Oriolo, 1994] in order to obtain the velocities $[v, \omega]^T$ from these forces.

An interesting feature of the simulator is the ability to dynamically change the values of controller parameters during the simulation. This allows the user to test different values and rapidly see the effects of the changes in the performance of the controllers. To be able to use this feature in his/her simulations, the user should implement a page in a property sheet containing the desired controller parameters, and use the values set directly into the controllers. Figure B.2 shows the parameters for the potential field controller. The user can set values to control the repulsion among robots, the goal attraction, etc.

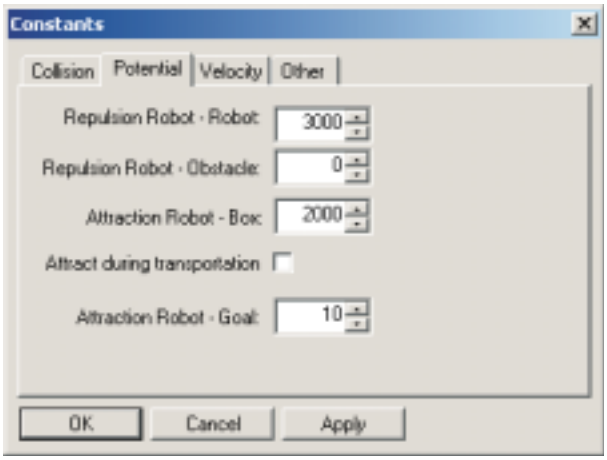


Figure B.2: Parameters for the potential field controller. The user can change the values dynamically.

B.6 Communication

Communication is fundamental for the majority of cooperative applications. In the simulator, the robots can use both implicit and explicit communication mechanisms to exchange information. Implicit communication does not require any special mechanism. The robots simply use their sensors in order to observe the actions of their teammates gathering the desired information. One example of the use of the simulator with implicit communication in a cooperative manipulation task is presented in Chapter 5 and detailed in [Pereira et al., 2002].

For the explicit communication, a message passing system was implemented in the simulator. A given robot can broadcast messages to all robots or send messages to a specific one. Two types of explicit communication can be used in the simulator: synchronous and asynchronous. Both of them were explained in Chapter 3. The inter-robot explicit communication in the simulator was implemented using windows messages. Each robot class has a handler that is called when a message is received. This handler is implemented by the user as a virtual function and specifies the actions that should be performed for each type of received message. Another important feature is that delays and errors (noise) in the communication mechanism can be simulated using stochastic functions, shortening the gap between simulation and real world applications. In this case, the robots can use an acknowledgement / time-out mechanism in order to assure the proper delivery of messages.

B.7 Sensors, Localization, and Mapping

Each robot is equipped with a sensor that allows the detection of obstacles, robots, and other targets in a certain range. This sensor is represented by a circular region around the robot. The robot is able to detect obstacles, boxes, and other robots inside this range. This sensor can be thought of as, for example, an omnidirectional camera, such as the ones that have been used in some real mobile robots to construct local range maps, perform localization tasks, and navigate trough cluttered environments [Das et al., 2001]. We also

implemented a bumper that can detect contacts among the robots and other entities. As mentioned above, other types of sensors can be implemented in new robot classes.

The robots are also equipped with odometry sensors that measure their position and orientation. For now, a strong assumption in the simulator is that each robot knows its exact position in the environment and can accurately detect obstacles, *i.e.*, we are not considering errors in odometry and other measurements. In real robots, uncertainty in localization and sensing can be corrected using stochastic algorithms and other techniques. For example, a robot can estimate its position using probabilistic distributions with the help of features detected in the environment and estimates from other robots.

Knowing their position and using information acquired by the sensors, the robots can construct maps of the environment and plan trajectories in some of the tasks. The Grassfire Algorithm, also called Distance Transform Algorithm, has been used in some applications for planning in real time. In this algorithm, the area is divided into cells and a value is attributed to each cell according to its distance to the goal. The planner starts at the goal cell and propagates distances through free spaces. Cells containing obstacles receive higher values. After the distance transforms are generated, a path is planned from the start position to the goal through the cells with decreasing values (see [McKerrow, 1991] for a detailed explanation). This algorithm is used locally, and thus can generate non-optimal paths. In spite of that, it generates a path whenever it is possible and can also be used together with other navigation strategies. Figure B.3 shows an example of a map (distance transform) and a plan generated using the grassfire algorithm. The gray levels indicate the cell values (increasing from white to black), the rectangles are obstacles and the line is the planned path from a start position to the goal.

B.8 Examples

To present some of the features of MuRoS, we developed two sample cooperative scenarios using the simulator [Chaimowicz et al., 2001a]. The first is a foraging task, in which a group of robots must search an environment looking for items, collect these items, and

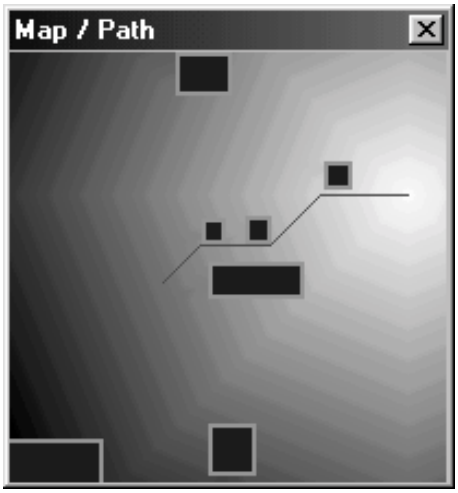


Figure B.3: Map and plan generated using the Grassfire Algorithm.

take them to a specific location. The second scenario is a cooperative manipulation where a group of robots grasp and push a box. They represent loosely and tightly coupled tasks respectively, being good testbeds for the simulator.

B.8.1 Foraging Task

The foraging task is a classical example of a loosely coupled cooperative task and has been used by several researchers as a testbed for simulations and real implementations of cooperative architectures, for example [Arkin et al., 1993] and [Drogoul and Ferber, 1992]. Using the simulator, we implemented different coordination methods with different levels of inter-robot communication and strategy, and analyzed completion time and speedup varying the number of robots.

In this simulation, the robots use distributed controllers and dynamic role assignment to control themselves during the execution of the task. Each robot can be in one of the following modes: Wander, Retrieve, and Transport (Figure B.4). In the Wander mode each robot searches the environment for items to be retrieved. When it detects an item, the robot changes its state to Retrieve and moves to get the item. After getting it, the robot

switches to the Transport mode and carries the item to the goal. While performing the Wander role, the robots move randomly in the environment. In the Retrieve and Transport modes, the robots use the artificial potential field method as explained in previous sections. In the Retrieve mode each robot is attracted by the item and in the Transport mode it is attracted by the goal.

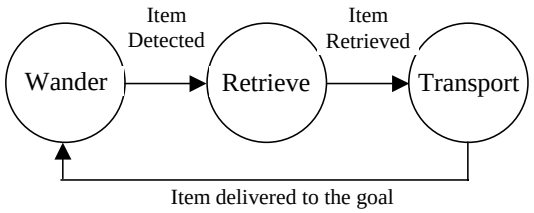


Figure B.4: Control modes and discrete switches for the forage task.

Figure B.5 shows a snapshot of the simulator during the forage task: the items to be collected are the small dots in the screen, while the robots are the circles. The sensor range of each robot is shown as a dashed circle around it. In this figure, there are four robots in the Transport mode carrying items (black), four in the Wander mode (light gray), and two in the Retrieve mode (dark gray, at the bottom-right of the screen). The goal is marked with an **x**.

Four different algorithms have been implemented to coordinate the robots during the execution of forage task:

1. **Random**: in this algorithm, the robots move randomly around the area and, when they find a item, they retrieve and transport it to the goal. There is no communication or multi-robot strategy.

2. **List**: this algorithm is very similar to the previous one, with the difference that each robot keeps a list with the position of the items that it has already detected but has not been able to retrieve because it can only transport one item at time.

3. **Communication**: in this algorithm, the robots also move randomly and maintain

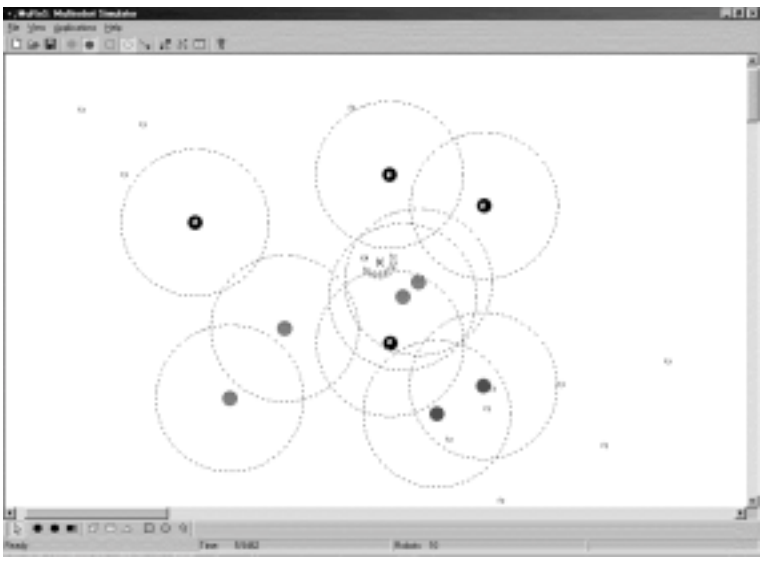


Figure B.5: Snapshot of MuRoS during the simulation of a forage task.

a list with the detected items, but they also exchange their lists using explicit communication. In this way, robots that are performing the Wander role can reallocate and retrieve items detected by other robots.

4. **Divide and Conquer**: in the divide and conquer algorithm, the robots do not communicate and, instead of moving randomly, they divide the search space among them and perform an exhaustive search in their area. When a robot finishes searching its area, it can start searching other areas.

Experiments were executed varying the number of robots (from 5 to 25) and the coordination algorithm. Each experiment was repeated 100 times and the average time to complete the task was computed. In these experiments, we used holonomic robots, 50 items and a search area of 10x10 meters with the goal placed in the middle. Each robot has a diameter of 30 centimeters and cannot carry more than one item at a time. Also, as mentioned, we consider that there is no uncertainty in the localization: the robots know their exact position and the position of the goal. The results are shown in the graph of Figure B.6.

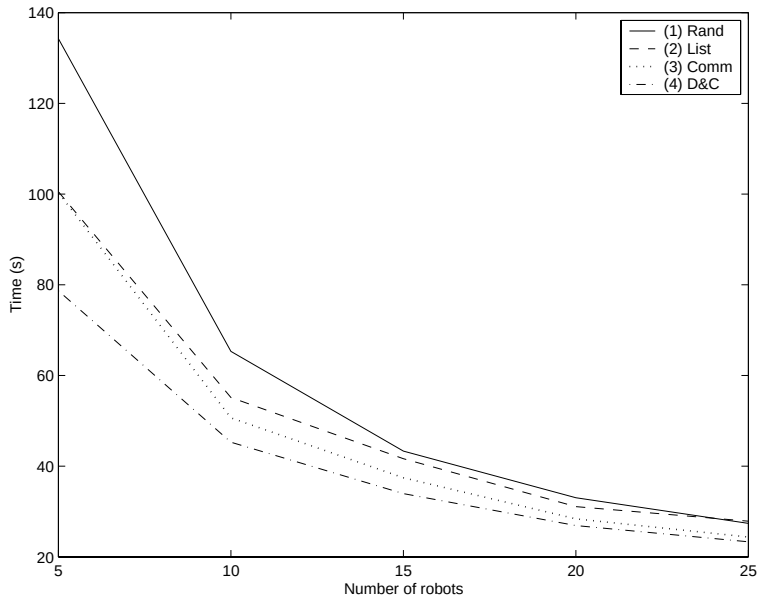


Figure B.6: Execution time × number of robots for the forage task.

When the number of robots is small, the Divide and Conquer algorithm has the best performance, showing that strategy is more important than keeping a list or communicating at this point. But the other two algorithms also have good performances when compared to the completely random approach. When the number of robots is increased, the difference in the execution time of all the methods decreases, showing that a larger number of robots brings benefits independently of the algorithm that is being used. There are two important results to point out when 25 robots are used: first, the performance of the Random algorithm is equal to the List algorithm, showing that with a large number of robots, keeping a list of the previously detected items does not contribute much to the task; second, the Communication and Divide and Conquer algorithms also have very similar performances, demonstrating that the use of communication is as important as strategy when a large number of robots is used.

A metric that can be used for this application is the speedup, a common metric used in the parallel programming community to analyze the benefits of using multiple processors

	5	10	15	20	25
Rand	4.72	9.69	14.62	19.16	23.13
List	4.99	9.09	12.03	16.13	17.98
Comm	5.00	9.89	13.38	17.65	20.55
D&C	4.50	7.81	10.41	13.13	15.16

Table B.1: Speedup results for the forage task.

or machines to perform a task. In our application domain, the speedup can be defined as:

$$Speedup = \frac{Execution\ time\ using\ 1\ robot}{Execution\ time\ using\ N\ robots}.$$

Table B.1 shows the speedup values computed for the four algorithms varying the number of robots. All the algorithms have large speedups, mainly the random and communication algorithms. The speedup shows how much the execution time is reduced when more than one robot is used. Speedup values close to n (n = *number of robots*) are called linear speedup and indicate that the task is benefiting totally from the use of multiple robots. This happens with the Communication Algorithm for $n \leq 10$ and the Random for $n \geq 5$. The communication helps orienting the wandering robots to the items that have been found by others, bringing significant improvements over the single robot approach. In the Random Algorithm, linear speedups were expected because the robots act in a completely independent fashion, thus, the addition of more robots causes a proportional reduction in the execution time. The Divide and Conquer is the algorithm that has the smallest speedup results because the use of strategy is effective even when only one robot is in use and the addition of new robots does not cause a linear improvement on the performance.

B.8.2 Cooperative Manipulation

Differently from the simulations that were presented in Chapter 5, the coordination of cooperative manipulation here is relatively simple: we don't have any leader-follower hierarchy and, consequently, role reallocations and exchanges are not necessary. The main

objective here is to demonstrate some of the features of MuRoS.

In this simulation, our approach to cooperative manipulation uses potential field controllers to guide the robots, and role allocations and communication for coordination. This is similar to the approach used in loosely coupled cooperation, with the main difference that communication and strict coordination are completely necessary. Basically, the robots are attracted by the box and the goal, and repelled by each other. The contacts with the box, obstacles, and other robots are computed using rigid body dynamic models [Song et al., 2001]. Figure B.7 shows the discrete modes and the transitions during the execution of the task. The transitions marked with a * are triggered by messages received from the other robots. Using a more formal definition, these transitions have guards that depend on variables set by explicit communication (global information).

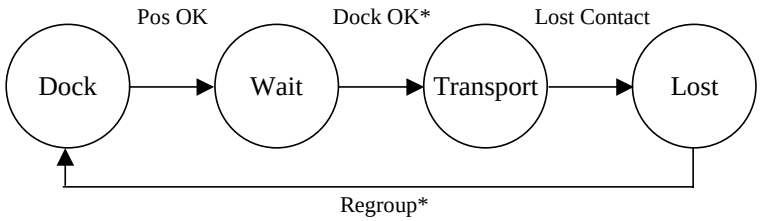
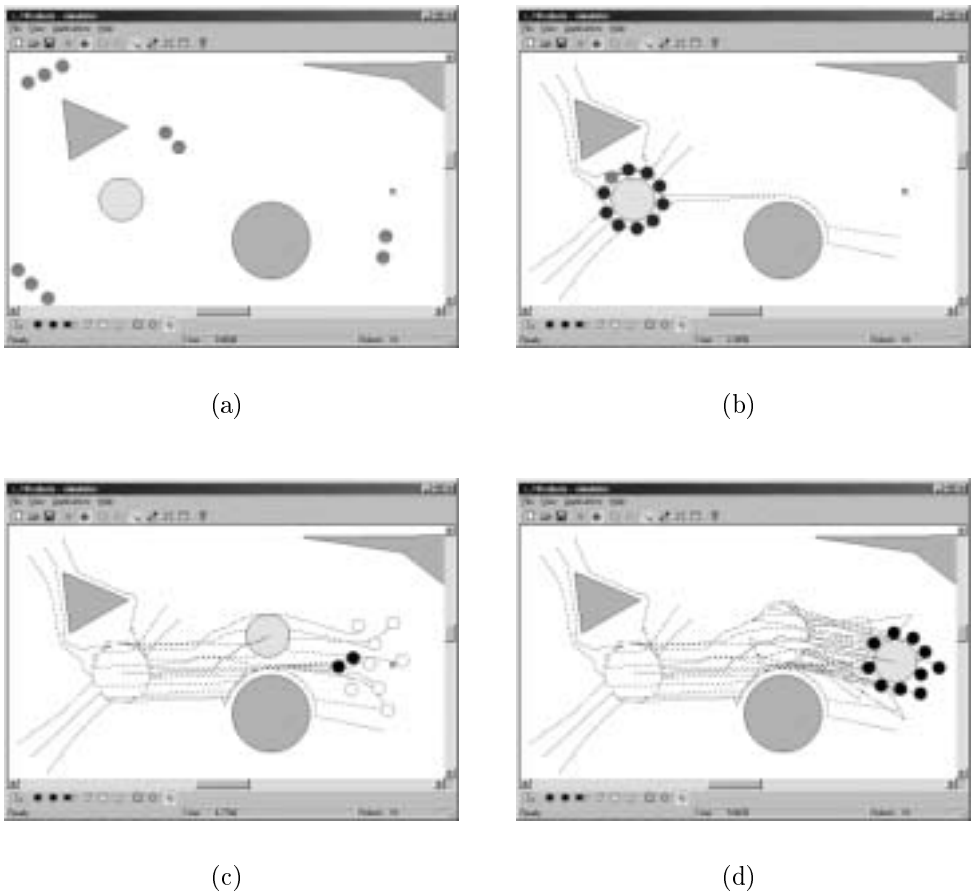


Figure B.7: Simplified discrete state diagram for the cooperative manipulation task.

Initially, the robots are in the Dock mode and are attracted by the box. At the same time, they are repelled by each other, being able to distribute themselves along the box and prepare for transportation. When one robot senses that it is close enough to the box, it switches to the Wait mode, and broadcasts a message communicating that it is ready. When all robots are ready, they switch to the Transport mode and there is a controller switch so that each robot becomes attracted by the goal. If for some reason one robot loses contact with the box, it will switch to the Lost mode, broadcast a message, and stop moving. If all robots lose contact, they regroup and start docking again.

Figure B.8 shows some snapshots of the simulator during the manipulation of a round box by ten holonomic robots in an environment with three obstacles. The goal position

is marked with an **x**. Snapshot (a) shows the robots in the Dock mode, starting to move in the direction of the box (light gray circle). Snapshot (b) shows nine robots in the Wait mode while the last one is still finishing the Dock phase. In (c) eight of the robots have lost contact with the box because of a collision with an obstacle. It is important to note that the robots lose contact when the box is outside their sensor range, so two robots (shown in black) are still in contact. As these two robots move towards the goal, they will also lose contact with the box. When this happens, they regroup, grab the box again and resume the transport, finishing the task (snapshot (d)).



(a) (b)

(c) (d)

Figure B.8: Snapshots of the manipulation task: (a) Robots (small circles) in the Dock mode. (b) Robots preparing to transport the box. (c) Robots lose contact with the box after colliding with an obstacles. (d) Transportation is finished after a robot regrouping.

Differently from loosely coupled tasks, communication and coordination are necessary in this task. Without strict coordination and communication the robots would not be able to transport the box and recover from failures. Also, depending on the size, shape, and mass of the box, a single robot alone or a small team cannot be able to complete the task. Some experiments were performed varying the mass and size of the box and in all cases at least 3 robots were necessary to transport the box. In the case of Figure B.8, eight robots were necessary to complete the task adequately.