Universitat de Girona

A MULTI-AGENT ARCHITECTURE WITH DISTRIBUTED COORDINATION FOR AN AUTONOMOUS ROBOT

Bianca Mariela INNOCENTI BADANO

ISBN: 978-84-692-1444-2
Dipòsit legal: GI-128-2009

UNIVERSITAT DE GIRONA

ESCOLA POLITÈCNICA SUPERIOR

A Multi-agent Architecture with Distributed Coordination for an Autonomous Robot

by

Bianca M. Innocenti Badano[1]

Advisors

Dra. Beatriz López[1] and **Dr. Joaquim Salvi**[2]

[1] Departament d'Enginyeria Elèctrica, Electrònica i Automàtica
[2] Departament d'Arquitectura i Tecnologia de Computadors
Universitat de Girona
Campus Montilivi - Edifici PIV, E- 17071 Girona

October 2008

A Multi-agent Architecture with Distributed Coordination for an Autonomous Robot

A thesis submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy at the Universitat de Girona

Author:

Bianca Innocenti Badano

Advisor:

Dra. Beatriz López

Advisor:

Dr. Joaquim Salvi

Programa de Doctorat: Enginyeria en Informàtica Industrial - Tecnologies Avançades de Control

Departament d'Electrònica, Informàtica i Automàtica

To Carles

Abstract

One of the challenges of robotics is to develop a robot control system capable of obtaining intelligent, suitable responses to changing environments. The basic requirement for achieving this aim is a robot control architecture and a hardware platform that can adapt the software and hardware to the current environment and situation. This has led researchers to design control architecture composed of distributed, independent and asynchronous behaviors.

Currently, researchers are introducing multi-agent technology into the control architecture because the properties of systems of this kind help to cover several of the features desired for the architecture.

In line with this research, this work proposes a new mobile robot control architecture with distributed coordination. This coordination mechanism is required to share the robot resources and is defined locally between the agents with conflicting resource requirements, instead of following a centralized approach as in the previous state-of-the-art architectures.

The proposed distributed coordination methodology consists of two steps: the first determines which agent wins the resource and the second how the resource exchange is carried out. The former is based on a private utility computation that is communicated between agents sharing resources. The second step ensures that the resource exchange among the different agents is performed smoothly and avoids abrupt robot behaviors.

This architecture has been conceived in such a way that the introduction of new hardware and software components is easy and does not result in the need for modifications of the existing agents. To make the introduction of new agents easy, as agent design pattern has been defined in order to define the agents' common features. This pattern has also lead to the development of a modular architecture inside the agent that allows separation of the different methods of achieving goals, collaboration and competition as well as resource coordination.

Another important feature of the proposed architecture is that agents can be coded in different programming languages and can run in different computers. An ad-hoc platform has been implemented in order to cope with real-time constraints and decentralization of the architecture, and a TCP/IP based communication protocol has been defined to establish message passing among agents. Also, a set of back agents has been defined to ensure the correct functioning of the whole system: robot control architecture and the multi-agent platform.

The multi-agent control architecture with distributed coordination has been tested on a real robot platform, a MobileRobots Inc. Pioneer 2DX robot. Several experiments have been carried out to test the architecture and the results show that the proposed features of the architecture have been accomplished.

Resum

Un dels reptes de la robòtica és desenvolupar un sistema de control per a un robot capaç d'obtenir respostes intel·ligents i adequades a entorns canviants. La base per aconseguir aquest propòsit és tenir una arquitectura de control per al robot i una plataforma hardware, que puguin adaptar el programari i el maquinari a l'entorn i la situació actual. Aquest fet ha portat als investigadors a dissenyar arquitectures de control formades per comportaments asíncrons, independents i distribuïts.

En l'actualitat els investigadors estan introduint la tecnologia multi-agent en les arquitectures de control ja que les propietats d'aquesta classe de sistemes ajuden a cobrir molts aspectes de les característiques desitjades per l'arquitectura.

En aquesta línea de recerca, aquest treball proposa una nova arquitectura de control per a un robot mòbil amb coordinació distribuïda. El mecanisme de coordinació necessàri per compartir els recursos del robot, es defineix localment entre els agents que presenten conflictes en l'ús del recurs, en lloc de fer servir una aproximació centralitzada com en les arquitectures prèvies de l'estat de l'art.

La metodologia de coordinació distribuïda proposada es fa en dos passos: el primer on es determina qui guanya el recurs i el segon on s'especifica com es fa l'intercanvi del recurs. El primer es basa en el càlcul privat de l'utilitat que es comunica entre els agents que comparteixen el recurs. El segon pas assegura que el canvi del recurs entre els diferents agents es faci de manera suau, evitant comportaments abruptes del robot.

Aquesta arquitectura ha estat concebuda de manera que introduir maquinari o programari nou sigui fàcil i que no impliqui haver de modificar els agents existents. Per fer fàcil la introducció dels agents nous, s'ha fixat un patró de disseny d'agents (agent's design pattern) per definir les característiques comunes dels agents. Aquest patró ha portat també, al desenvolupament d'una arquitectura modular dins l'agent que permet separar els diferents mètodes d'obtenció dels objectius, de col·laboració i de competició així com la coordinació dels recursos.

Una altra característica important de l'arquitectura proposada és que els agents poden escriure's en diferents llenguatges de programació i poden executar-se en diferents ordinadors. S'ha implementat una plataforma ad-hoc per tenir en compte l'operació en temps real i la descentralització de l'arquitectura, i s'ha definit un protocol de comunicació basat en TCP/IP per establir el pas de missatges entre els agents. A més, s'ha definit un conjunt d'agents de suport (back agents) per assegurar el funcionament correcte de tot el sistema, tant de l'arquitectura de control del robot com de la plataforma multi-agent.

L'arquitectura de control multi-agent amb coordinació distribuïda ha estat provada en un robot real, un Pioneer 2DX de la casa MobileRobots Inc. S'han fet varis experiments per examinar l'arquitectura. Els resultats mostren que les característiques proposades per l'arquitectura s'han complit.

Resumen

Uno de los retos de la robótica es desarrollar un sistema de control para un robot capaz de obtener respuestas inteligentes y adecuadas a entornos cambiantes. La base para conseguir este propósito es tener una arquitectura de control para un robot y una plataforma hardware, que puedan adaptar el software y el hardware al entorno y a la situación actual. Este hecho ha llevado a los investigadores a diseñar arquitecturas de control formadas por comportamientos asíncronos, independientes y distribuidos.

En la actualidad los investigadores están introduciendo la tecnología multi-agente en las arquitecturas de control porque las propiedades de esta clase de sistema ayudan a cubrir muchos aspectos de las características deseadas para la arquitectura.

En esta línea de investigación, este trabajo propone una nueva arquitectura de control para un robot móvil con coordinación distribuida. Este mecanismo de coordinación necesario para compartir los recursos del robot, se define localmente entre los agentes que presentan conflictos en el uso del recurso, en lugar de utilizar una aproximación centralizada como en las arquitecturas previas del estado del arte.

La metodología de coordinación distribuida propuesta se realiza en dos pasos: el primero donde se determina quién gana el recurso y el segundo donde se especifica cómo se hace el intercambio del recurso. El primero se basa en el cálculo privado de la utilidad que se comunica entre los agentes que comparten el recurso. El segundo paso asegura que el cambio del recurso entre los diferentes agentes se haga de manera suave, evitando comportamientos abruptos del robot.

Esta arquitectura se ha concebido de manera que introducir software o hardware nuevo sea fácil i que no implique tener que modificar los agentes existentes. Para facilitar la introducción de los nuevos agentes, se ha definido un patrón de diseño de los agentes (agent's design pattern), para definir las características comunes de los agentes. Este patrón ha llevado, también, al desarrollo de una arquitectura modular dentro del agente, que permite separar los diferentes métodos de obtención de los objetivos, de colaboración y de competición así como la coordinación de los recursos.

Otra característica importante de la arquitectura propuesta, es que los agentes pueden escribirse en diferentes lenguajes de programación y ejecutarse en distintos ordenadores. Se ha implementado una plataforma ad-hoc para considerar la operación en tiempo real y la descentralización de la arquitectura, y se ha definido también, un protocolo de comunicación basado en TCP/IP para establecer el paso de mensajes entre los agentes. Además, se ha definido un conjunto de agentes de soporte (back agents) para asegurar el correcto funcionamiento de todo el sistema: la arquitectura de control del robot y la plataforma multi-agente.

La arquitectura de control multi-agente con coordinación distribuida se ha probado en un robot real, un Pioneer 2DX de la casa MobileRobots Inc. Se han hecho varios experimentos para testear la arquitectura. Los resultados muestran que las características propuestas para la arquitectura se han cumplido.

Acknowledgements

I would like to thank my thesis directors for their belief in me and encouraging me during the almost 7 years that this thesis has been in the making. Thanks to Dr. Beatriz López for her patience, support and most beneficial advice and to Dr. Joaquim Salvi for his valuable comments and time spent reading and correcting the new ideas that have materialized in papers and reports.

I also would like to thank to all those people that have contributed in some way to the development and completion of this thesis. Thanks to Dr. Dídac Busquets for listening and suggesting interesting ideas to improve it; to Josep Tomàs and Marc Rodríguez for helping me to solve the Grill hardware problems (the pioneer robot) and carry out the real experiments; to Toni Verdú and Ingrid Azorín for maintaining my PC in good condition and for the re-installation and hang-up solutions provided in record time; to Dr. Josep Forest who helped me to choose the PC for the robot, including the operating system, and for introducing me into the Gentoo world (and repairing my stupid mistakes with this software); to Dr. Pere Ridao, Dr. Narcís Gascons, Dr. Edin Omerdic and Andrés El-Fakdi for helping me through the universe of robot modeling and identification; to the EXIT group for their support and encouragement; to Silvia Solé, Anna Renart, Rosa Teixidor, Montse Vila and Isela Ibrahimovic for their assistance with the paper work and the long conversations about ... everything; and to the students that helped in the development of the software and in the implementation of some of the methods presented in this thesis.

I acknowledge my friends that were always there, in good and bad times. Thanks to Sonia, Mar and Isela for listening to all my complaints, problems and excuses and for maintaining our friendship with little help from me.

Finally I would like to thank my family: my parents for being always by my side, for giving me the opportunity of having a career and for supporting my decision to move to Spain and marry there; my sister and brother for sharing my life and filling it with memorable experiences and love; my in-laws for accepting me into their lives and taking care of me as if I was another daughter and sister; and last but not least my husband Carles, whose cleverness, temperament, patience and good-humor always helped and most especially when times were desperate, with his suggestions and new ideas about how to carry on (even though some of them seemed completely crazy at first): thanks for being there and for giving light and happiness to my life.

This work has been partially supported by Spanish government projects DPI2002-04018-C02-02, TIN2004-06354-C02-02 and DPI2006-09370, the AGAUR project 2005SGR 00296 and the AECI Intercampus project A/1562/04, which are hereby gratefully acknowledge. I would also like to thank the PLANET European network and AgentCities.

Contents

List of Figures

List of Tables

List of Acronyms and Abbreviations

AA : ARTIS Agent

ACC : Agent Communication Channel

ACL : Agent Communication Language

ACTRESS : ACTor-based Robot and Equipments Synthetic System

ADE : APOC Development Environment

AI : Artificial Intelligence

ALFA : A Language for Action

AMS : Agent Management Services

APOC : Activating, Processing and Observing Components

ARA : Autonomous Mobile Robot Agent Architecture

ARCoS : Autonomous Robot Control System

ARIA : Advance Robotics Interface for Applications

ARMADiCo : Autonomous Robot Multi-agent Architecture with Distributed Coordination

ATLANTIS : A Three-Layer Architecture for Navigating Through Intricate Situations

AuRA : Autonomous Robots Architecture

BERRA : Behavior-based Robot Research Architecture

CARMEN : Carnegie Mellon Robot Navigation Toolkit

CEBOT : Cellular roBOTics system

CLARAty : Coupled Layered Architecture for Robotic Autonomy

CORBA : Common Object Requesting Broker Architecture

CPU : Central Processing Unit

CVM : Curvature Velocity Method

DAMN : Distributed Architecture for Mobile Navigation

DC : Direct Current

DF : Directory Management Services - Directory Facilitator

DPAA : Dynamical Physical Agents Architecture

EA : Elementary Agent

FIPA : Foundation for Intelligent Physical Agents

FO : Final Orientation

GUI : Graphical User Interface

HA : High-level Agent

HRI : Human Robot Interface

ICL : Interagent Communication Language

IDEA : Intelligent Distributed Execution Architecture

IEEE : Institute of Electrical and Electronics Engineers

IPC : Inter-Process Communication

IS : Intelligent Server

L-ALLIANCE : Learning ALLIANCE

LS : Last Square

MARIE : Mobile and Autonomous Robotics Integration Environment

MAS : Multi-Agent System

MaSE : Multi-agent System Engineering

Miro : Middleware for Robots

MPA : Manager Platform Agent

NaT : Navigation Templates

O2CA2 : Object Oriented Control Architecture for Autonomy

OAA : Open Agent Architecture

P : Precision

P/S : Player/Stage

P2OS : Pionner 2 Operating Sytem

PC : Personal Computer

PID : Proportional Integral Derivative

POMDP : Partially Observable Markov Decision Process

POSINI : Initial Position

PRBS : Pseudorandom Binary Sequence

PRS : Procedural Reasoning System

Pyro : Python Robotic

RA : Robot Agent

RA : Remote Agent

RAP : Reactive Action Packages

RDE : Robotic Development Environments

RMI : Remote Method Invocation

ROCI : Remote Objects Control Interface

RPC : Remote Procedure Call

RTMAS : Real-Time Multi-Agent Systems

SLAM : Self Localization and Mapping

SONREAD : Sonar Readings

SSS : Servo, Subsumption and Symbolic Architecture

TA : Total Time the Avoid Agent has control

TCA : Task Control Architecture

TD : Traveled Distance

TES : Task Execution Supervisor

TG : Total Time the Goto Agent has control

TGt : Total Time the Gothtough Agent has control

TPC/IP : Transmission Control Protocol/Internet Protocol

TT : Total Time

UA : User Agent

UML : Unified Modeling Language

VO : Virtual Operator

VOMAS : Virtual Operator Multi-Agent System

CHAPTER 1
Introduction

"How old are you?" she wanted to know. "Thirty-two," I said. "Then you don't remember a world without robots.[...] To you, a robot is a robot. Gears and metal; electricity and positrons. Mind and iron! Human-made! If necessary, human-destroyed! But you haven't worked with them, so you don't know them. They're a cleaner better breed than we are."

From *I, Robot* by Isaac Asimov

In this chapter the goals of the thesis will be enumerated, some related concepts introduced and, finally, the outline of the dissertation presented.

1.1 INTRODUCTION

Artificial Intelligence Robotics concentrates on how a mobile robot can handle unpredictable events in an unstructured world (as opposed to Industrial Robotics, which is concerned with a robot's dynamics and kinematics [89]). With this as their aim, there are researchers who are involved in a long term effort to integrate perception, navigation, planning and uncertainty management methods in a single robot architecture. Traditionally, most researchers has focused on a module based approach, in which each robot component is implemented in a module [13, 3, 111].

The design of an autonomous robot is highly complex, since a robot has to integrate cognitive abilities with other capabilities such as locomotion, prehension, and manipulation [71]. Each of these robot capabilities involves different aspects of intelligence, and different intelligent tools have be used consistently to implement them.

More recently, several researchers have concerned themselves with the significant impact of agent technology on the world of robotics [61, 60]. The majority of multi-agent robot system architectures, rather than being built to control single robot systems, are intended for multi-robot systems ([54, 121, 36, 31]). The mapping from a robot to an agent seems straightforward, as long as each robot represents a physical entity with a specific task. Such mapping, however, can also be extended inside a robot, to carry out different activities and to respond in time to the environment. For example, some activities might require input from a variety of sensors, to control the motion of the robot's wheels, and plan paths. In addition the robot needs to avoid obstacles and deal with uncertainty due to sensor constraints, power levels and other events that it may encounter. Such activities have to be organized in order for the robot's goals to be achieved, and multi-agent methodologies offer an appropriate background.

Using a multi-agent approach, the robot's architecture can be decomposed into flexible autonomous subsystems (agents). The architecture can then be described at a higher level,

defining the agents that have to be in the system, the role of each of them, the interactions among them, the actions each of them performs, and the resources they need. Since the multi-agent system is inherently multi-threaded, each agent has its own thread of control; each agent decides whether or not to perform an action at the request of another agent (autonomy); agents establish agreements among themselves, while keeping their autonomy sharing their knowledge and acting together to accomplish specific common goals.

Agents need to interact to coordinate their activities so that control of the robot is achieved. All of those processes, the agent's own decision making, interaction and coordination need to be done under real-time constraints. This poses certain challenges for current multi-agent approaches, which this Ph.D. thesis addresses while at the same time proposing a new architecture for mobile robots.

1.2 Related Fields

In this section the basic concepts of robotics, agents and multi-agent fields are presented. Most of the concepts connected with this thesis have, in fact, been extensively dealt with in the literature. Nevertheless, some definitions are presented below in order to further clarify their intended meaning.

1.2.1 Robots

The word *robot* was introduced in 1921 by Czech writer Karel Capek in his play R.U.R (Rossum's Universal Robots), as the name for the artificial people created by an unseen inventor (Rossum) to replace humans in any job. Capek described these artificial people as robots, a word derived from the Czech word *robota* meaning literally serf labor. Robots were constructed to serve and to free humans from any type of labor [3].

This concept has evolved over time and several definitions for the word now exist in the literature, but for the purposes of this thesis, an *an autonomous robot* is understood to be *an intelligent machine capable of performing tasks in the outside world by itself, without any explicit human control over its movements* [7]. Also, an *intelligent machine* is taken to be *a machine able to extract information from its environment and use knowledge about its world to move safely in a meaningful and purposive manner* [3]. That means a machine that can perceive and act in its environment, taking appropriate decisions over a period of time.

To provide the robot with these capabilities, a robot control architecture is necessary. A robot control architecture is defined as a mapping of sensory information into actions in the real world, in order to accomplish a certain task. It is a way of integrating different kinds of hardware and software modules. Furthermore, it plays an important role in maintenance tasks and in the addition of new modules. The three paradigms of robot control architecture as well as some of the most important architectures are explained in Chapter 2.

Throughout this Ph.D. thesis, the word robot will refer to mobile robots only.

1.2.2 Agents and Multi-agents Systems

There is no universally accepted definition of the term agent, but the one adopted in this work is the one presented in [133]:

An agent is a computer system that is situated in an environment and that is capable of autonomous action in this environment in order to meet its design objectives (see figure 1.1).

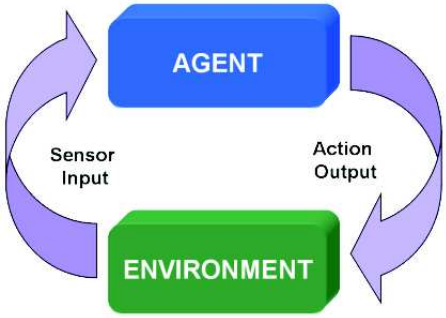


Figure 1.1: An agent in its environment.

This is a global definition; many things can be thought of as agents that certainly do not involve ”intelligence”. *An intelligent agent is an agent that is capable of flexible autonomous action in order to meet its design objectives*, where flexibility means three things [133, 131, 56]:

- ***Reactivity:*** intelligent agents are able to perceive their environment, and respond in a timely fashion to changes that occur in it in order to satisfy their design objectives. A *reactive* system is one that maintains an ongoing interaction with its environment, and responds to changes that occur in it (in time for the response to be useful).
- ***Pro-activeness:*** intelligent agents are able to exhibit goal-directed behavior by *taking the initiative* in order to satisfy their design objectives.
- ***Social ability:*** intelligent agents are capable of interacting with other agents (and possibly humans) in order to satisfy their design objectives. *Social ability* is the ability to interact with other agents (and possibly humans) via some kind of *agent - communication language*, and perhaps cooperate with others.

When problems become more complex, realistic and large-scale, they progress beyond the capabilities of a single agent. The only reasonable way to tackle this type of problems is to create an organization of agents in which each agent is highly specialized in solving a particular aspect of the problem. This organization is known as a Multi-Agent System (MAS). Formally defined, *a Multi-Agent System is a distributed system that contains a collection of agents that work together in order to solve problems* [19, 133, 87]. Agents in a MAS interact through communication.

Some significant advantages of a MAS are [87, 56, 19, 122, 112]:

- **Faster problem solving by exploiting parallelism:** resolution of problems too large for a single agent to handle because of resource limitations or the risk of a bottleneck in centralized systems or failure at critical times.

- **Easier legacy system integration:** allowing the interconnection and inter-operation of multiple existing legacy systems within a wider cooperating agent community in which they can be exploited by other pieces of software.
- **Natural design metaphor:** providing solutions to problems that can be regarded naturally as a society of autonomous interacting components-agents
- **Distribution of data and control:** efficient use of information sources that are spatially distributed or are of distributed expertise.
- **Enhanced performance of computational efficiency, reliability, extensibility, robustness, maintainability, responsiveness, flexibility and reuse.**

In accordance with these advantages, the multi-agent system is suitable for developing the control architecture of a robot since it has inherent characteristics that are also desirable for the architecture.

1.3 GOALS

The main goal of this work is to **develop a robot control architecture for a mobile robot that supports complex robot behaviors.**

Some of the desired features of such an architecture are flexibility, real-time response and coherent behavior. *Flexibility*, at both hardware and software levels, means that new sensors and actuators can be easily added and removed, the control architecture can be used in different robot platforms, and software components can be coded in various programming languages. *Real-time response* implies that the architecture can produce actions with time constraints. And *coherent behavior* means the results of the applied actions must be rational and smooth.

With these considerations in mind, the main goal can be subdivided into three more specific sub-goals:

- **Multi-agent control architecture:** design of the control architecture of the robot, based on a multi-agent system. The fact that the architecture is a multi-agent system provides flexibility in terms of the software level. This architecture will have reactive and deliberative agents at least. The reactive agents will guarantee that simple tasks are achieved under time constraints (real-time if necessary) while deliberative agents will grant planning and reasoning. The whole architecture must assure the safety of the robot and the environment, so it should provide the mechanisms to deal with hardware and software failures.
- **Coordination mechanisms:** providing the agents in the system with the appropriate coordination mechanisms in order to achieve rational behavior, avoiding non-feasible, unsecured and fortuitous actions that can provoke undesirable results in task achievements.
- **Experimentation with a mobile robot:** evaluation of the proposed architecture with real experiments to determine the feasibility and limitations of the proposed architecture, using a real platform.

1.4 Structure of the Thesis

The contents of the thesis are structured as follows:

- **Chapter 2:** *Review of Robot Control Architectures*. This chapter presents the different kinds of robot architectures and, following a selected criteria, overviews the most representative of them in order to reduce the wide number of architectures proposed in the literature. Several relevant features for a robot control architecture are identified, and all the systems analyzed are compared on the basis of these features.
- **Chapter 3:** *Autonomous Robot Multi-Agent Architecture with Distributed Coordination (ARMADiCo)*. This chapter describes the proposed robot control architecture based on a multi-agent system. The architecture has a distributed coordination mechanism that allows the agents to decide about resource coordination and management for themselves. In order to develop the architecture and to help progress through the different steps needed to obtain a functional architecture a scientific methodology has been followed; and an agent's design pattern has been used to define the different agents that make up the architecture.
- **Chapter 4:** *Experimental Setup*. This chapter details the multi-agent methodology, the multi-agent platform used to implement the multi-agent architecture, the robot and its dynamic model, and the implementation of the collaborative control in respect of the setting up of ARMADiCo.
- **Chapter 5:** *Experimental Results*. This chapter presents the results obtained with the multi-agent control architecture using simulation and a real robot. The collaborative control is tested in several scenarios. Agent interactions and distribute coordination are tested, the latter in two situations, using an abrupt and a smooth change of resources control. The difficulty of adding a new agent in the architecture is experimented with, too. Finally, the architecture is tested with a real robot.
- **Chapter 6:** *Conclusion and Future Work*. This chapter concludes the thesis by summarizing the work and presenting conclusions as well as recommendations for future work. It also enumerates the publications generated during this work.

CHAPTER 2

Review of Robot Control Architectures

The technology itself invited the behavior. Distributed agent systems ran by themselves. That was how they functioned. [...] Autonomy was the point of it all. [...] And they set the swarm free. [...] The swarm evolves on its own, the less successful agents die off, and the more successful agents reproduce the next generation. After ten or a hundred generations, the swarm evolves toward a best solution. An optimum solution.

From *Prey* by Michael Crichton

In this chapter, deliberative and reactive paradigms as well as the most representative architecture of both are first described. Next, hybrid architectures are explained. In particular, the focus is on the study of multi-agent architectures, the new software paradigm that has been used to develop hybrid architectures since the middle 90's (replacing modules), differentiating multi-agent architectures for a single robot (the focus of this thesis) from multi-agent architectures for multi-robot communities. Next, some robotic development environments are outlined, together with other related architectures. Finally a comparative analysis of the architectures cited is provided.

2.1 INTRODUCTION

In Chapter 1, the definition of what an intelligent robot is considered to be was given. This definition implies that the designer must organize in some way the intelligence of the robot. To accomplish this task, three paradigms are used: deliberative or hierarchical, reactive and hybrid.

The deliberative or hierarchical schema is the oldest and is based on traditional Artificial Intelligence. In this kind of architecture, each layer provides subgoals to the layer below. They include a global world model which is modified and updated through perception (sensory information). Based on this world model, planning and reasoning are carried out that result in actions to be performed by the robot. This means that the robot first senses, then thinks and finally moves. As reasoning takes a significant amount of time, it becomes a bottleneck in the architecture. In addition, the world model should be as detailed as possible

in order to permit classic planners to work with it. This is one of the main drawbacks of this kind of architecture, since the world model is application dependent and is very difficult to build and maintain. Another important disadvantage is the difficulty for the architecture in handling uncertainty (due to sensor noise, actuator errors, etc.).

In contrast to deliberative architectures, reactive and behavior-based architectures (originally proposed by Brooks in 1986) are characterized by a close coupling between perception and action. Instead of a layered layout, behavioral architectures have a horizontal decomposition, in which behaviors work in parallel, concurrently and asynchronously. Behaviors are inherently modular, easy to test in isolation from the system and support incremental expansion of the capabilities of a robot. In reactive architectures, sensing is local to each behavior (a local world model) and hence there is no global world model as in hierarchical architectures. One disadvantage of reactive architectures is they do not perform well when carrying out complex tasks (i.e. that need planning). Also, as they do not have a global world model, dead ends and repetitive movements can be reached when different behaviors take control of the robot.

Neither the purely reactive scheme (sense-act) nor the purely deliberative architectures (sense-plan-act paradigm) perform well when performing complex tasks, because of difficulties in modeling the world and relying too much on inadequate sensors [97].

Thus, hybrid architectures are used to control robots because they have both desirable properties: **reactivity**, so they can respond in real-time to changes in dynamic environments and **deliberation**, so they can plan actions ahead in time.

They use asynchronous processing techniques (multi-tasking, threads, etc.) to allow deliberative functions to be executed independently of reactive behaviors (a planner can be slowly computing the next goal for a robot to navigate to, while it is reactively navigating toward its current goal with fast update rates). Additionally, good software modularity allows subsystems or objects in hybrid architectures to be mixed and matched for specific applications [89].

There are many hybrid architectures for controlling a single robot. In order to choose the most representative ones for this study, the following criteria have been applied:

1. *The most well-known and most cited:* as there are a lot of mobile robot control architectures, the most cited in the literature have been chosen to represent the hybrid paradigm.

2. *Developed since 2002*: as most of the surveys in the literature are from before 2002, newer architectures have been encompassed, most specially those that are agent-based or multi-agent systems.

3. *Implemented and tested on real robots*: this criterion is the most important since several architectures have been tested only in simulated environments. Also, some have been developed as softbots or web-bots. Their implementation in real robots guarantees dealing with realistic situations, uncertainty, noise in sensors and errors in actuators.

Figure 2.1 summarizes the three paradigms used to develop robot control architecture. As can be seen, the hybrid paradigm, which has the advantages of the reactive and deliberative schemas, can be achieved using a modular-based or an agent-based approach in the architecture. In addition, multi-robot systems and other systems for controlling robots are included in this figure.

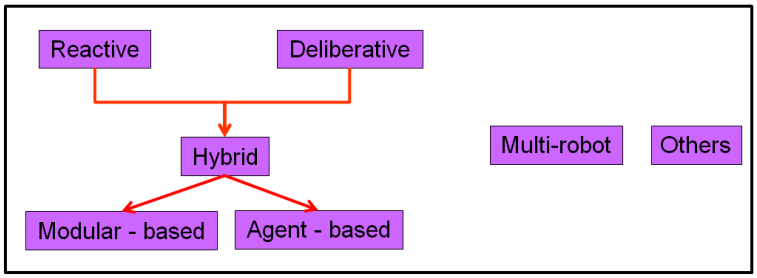


Figure 2.1: Robot Control Architecture.

2.2 Deliberative or Hierarchical Control Architectures

These types of architectures are structured in layers. Figure 2.2 shows the sequence they use to control robots. First the robot senses the world and constructs a global world map. Then, ”eyes” closed, it plans all the directives needed to reach the goal. Finally, the robot acts to carry out the first directive [89]. Each step is performed at the corresponding layer.

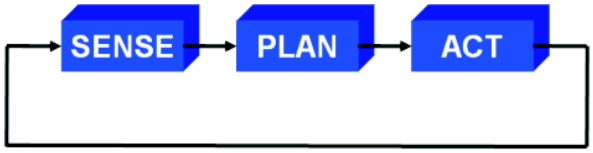


Figure 2.2: Hierarchical Architecture sequence.

Deliberative reasoning systems often have several common characteristics:

- They are hierarchical in structure with a clearly identifiable subdivision of functionality.
- Communication and control occurs in a predictable and predetermined manner, flowing up and down the hierarchy, with little if any lateral movement.
- Higher levels in the hierarchy provide subgoals for lower subordinate levels.
- Planning scope, both spatial and temporal, changes at the different layers in the hierarchy. Time requirements are shorter and spatial considerations are more local at the lower levels.
- They rely heavily on symbolic representation world models.
- Deliberative systems have a very structured communication orders.

A relatively complete knowledge about the world is required. This knowledge is used to predict the outcome of the robot’s actions, an ability that enables it to optimize its performance relative to its model of the world. Deliberative reasoning often requires strong assumptions about this world model first and foremost that the knowledge upon which the reasoning is based is consistent, reliable, and certain. If the information model is inaccurate or has changed since it was obtained, the outcome of the reasoning may err.

Thus, hierarchical control is seemingly well suited for structured and highly predictable environments, but is inappropriate for dynamic environments which require timely responses.

One of the most representative architecture in this paradigm is **RAP: Reactive Action Packages** [39]. This three-layered architecture is formed by planning, execution and control systems (see Figure 2.3).

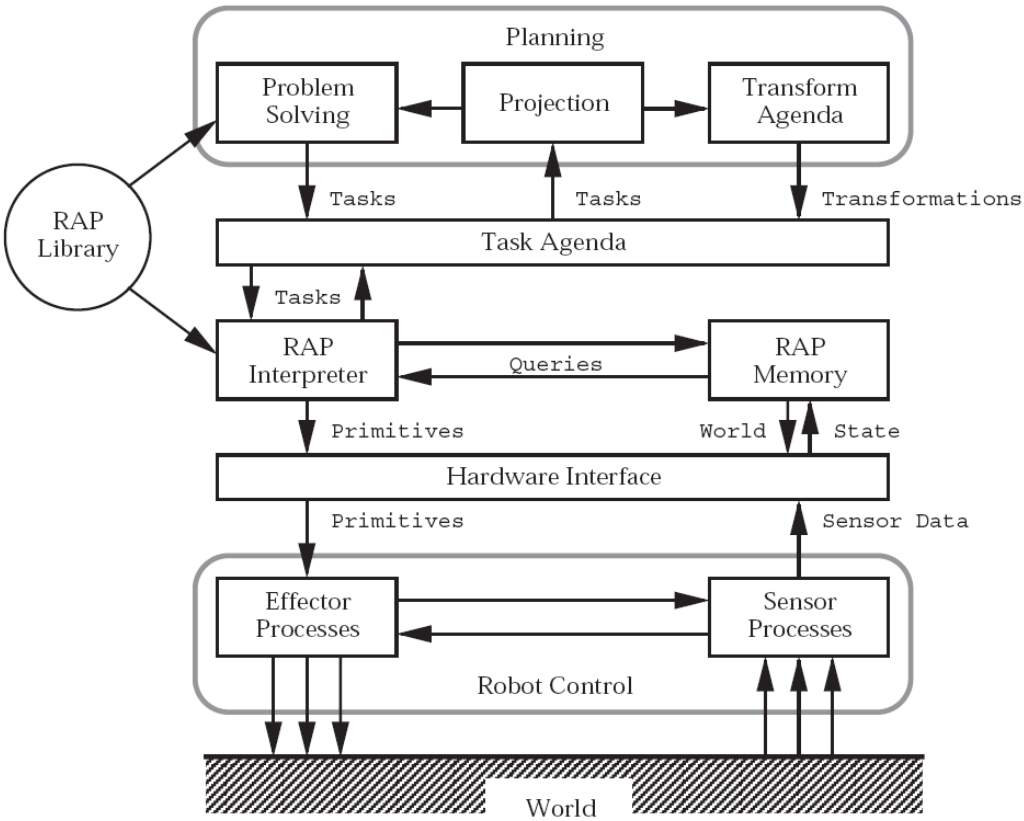


Figure 2.3: Layout of RAP architecture (extracted from [39]).

The planner decomposes the robot mission into ”sketchy”, high-level sequences of tasks that will be executed in order to achieve the goal. The execution system takes each task provided by the planner, and expands it into more detail. Finally the control system translates the tasks into low level commands that are handled by the robot's sensors and actuators. Each task contains a list of methods that allows the task to be performed and the conditions that must be satisfied to apply the methods. A method can be a primitive action or a list of sub-tasks to be installed in the task queue. The system works by successively expanding tasks in the queue until they either finish or fail. When a task fails, an alternate method is tried. RAP maintains a success check of the method applied and when it does not achieve the desired state, the RAP system tries another method. Only when all possible methods have been tried several times does RAP give up.

Another particularity of the RAP system is that it can deal with opportunities (situations in the world that cannot be predicted by the planning system) and emergencies (new facts that appear and that the system must respond to immediately, dropping the current task). This is done by allowing reaction tasks to exist on the execution agenda concurrently with tasks making up a plan, and assigning them priorities that allow the system to shift from one goal to another when appropriate. In this way some responsiveness to changing conditions in the world is achieved.

2.3 Reactive Control Architectures

An alternative to the deliberative paradigm is reactive architectures. Figure 2.4 shows the sequence used to control robots with this schema.

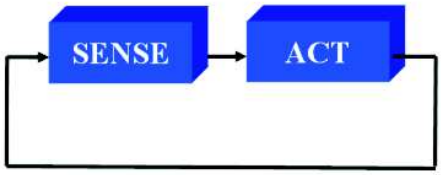


Figure 2.4: Reactive Architectures.

Reactive robotic systems have the following characteristics:

- Behaviors serve as the basic building blocks for robotic actions. A behavior in these systems typically consists of a simple sensor-motor pair, with the sensory activity providing the necessary information to satisfy the applicability of a particular low-level motor reflex response (stimulus-response).
- Use of explicit abstract representational knowledge is avoided in the generation of a response. Purely reactive systems react directly to the world as it is sensed, avoiding the need for intervening abstract representational knowledge. There is no planning ahead or internal state information.
- Animal models of behavior often serve as a basis for these systems.
- These systems are inherently modular from a software design perspective and they are also inherently concurrent.
- Reactive systems are very fast in terms of computations and motions.
- Purely reactive architecture is unable to learn.

Several researchers [79, 104] agree in considering a behavior-based paradigm as different from a reactive paradigm, and some would also add free market and immunity based architectures to the previous list [118].

Reactive robotic systems serve best when the real world cannot be accurately characterized or modeled. Very often, uncertainty, unpredictability and noise from the world cannot be removed. Reactive architectures were developed in response to this difficulty and choose instead to deal with these issues from the beginning, relying heavily on sensing without constructing potentially erroneous global world models.

One of the most representative architectures in this paradigm is **Subsumption Architecture** [13] (see Figure 2.5). This architecture is a composition of hardware implementations with a number of levels of competence that are ”an informal specification of a desired class of behaviors for a robot over all the environments it will encounter” [13]. The higher the level of competence is, the more specific the desired class of behaviors will be.

The levels of competence allow layers of control systems for each of them to be built and new layers to the existing set of levels of competence to be added so that the next highest level of overall competence can be achieved. In this organization of behaviors there is

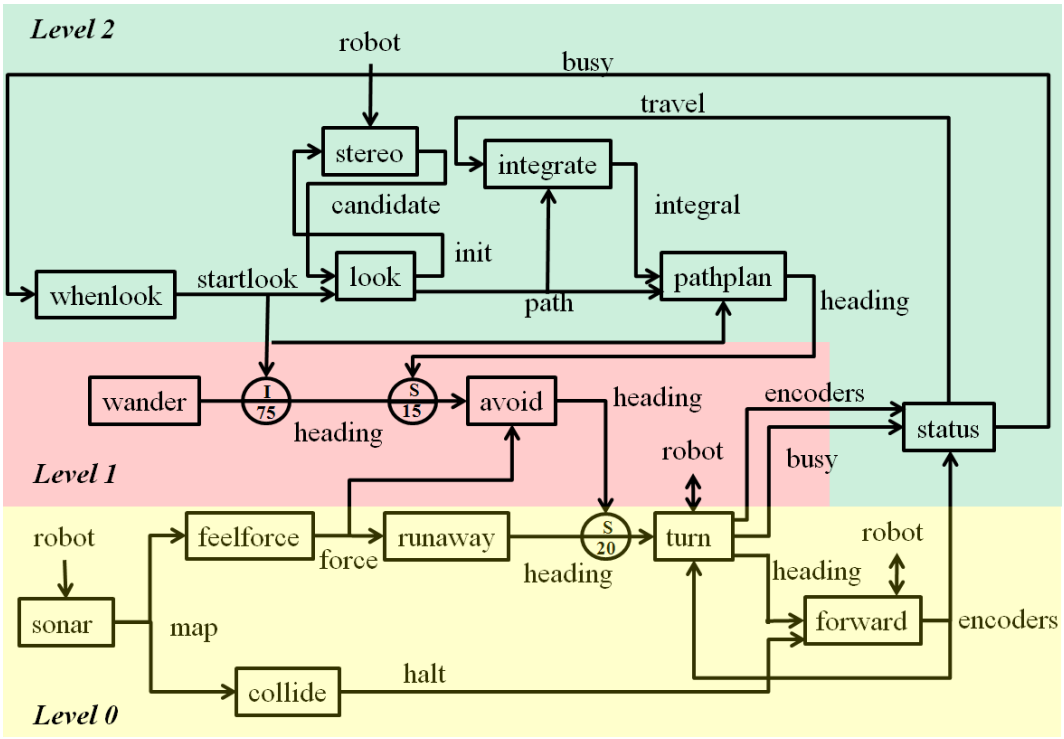


Figure 2.5: Layout of Subsumption architecture (extracted from [13]).

a hierarchy in which lower level behaviors have no awareness of higher layers; complex behaviors subsume simple ones.

Layers are built from a set of small asynchronous processors (finite state machines) that send messages to each other. Communication among finite state machines is achieved through connecting wires (as inputs and outputs of the processors). In Figure 2.5 these processors (squares) can be seen, as well as the inputs and the outputs. Input signals can be suppressed and replaced with the suppressing signal (circles with a capital letter *S*). Output signals can be inhibited (circles with a capital letter *I*). Inhibition and suppression are the two mechanisms used to coordinate behaviors. In the former signals are not transmitted along the augmented finite state machine and thus do not reach actuators, while in the latter signals are replaced with suppressing messages.

2.4 Hybrid Control Architectures

Nowadays the robotic community believes that Hybrid Deliberative/Reactive architectures, if done properly, can potentially yield the best of both worlds: the use of abstract representational knowledge of purely hierarchical systems, and the responsiveness, robustness and flexibility of purely reactive systems.

Hybrid architectures permit reconfiguration of reactive control systems based on available world knowledge through their ability to reason about underlying behavioral components. Figure 2.6 shows the sequence used to control robots in hybrid architectures.

The sequence implies that, the robot first plans how to accomplish a mission or a task based on a global world model, and then activates a set of behaviors to fulfill the plan that

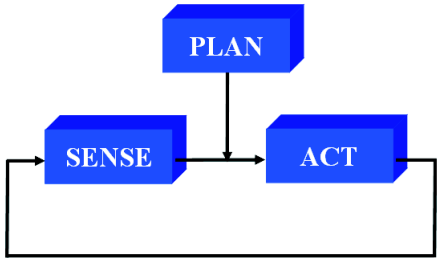


Figure 2.6: Hybrid Architecture sequence.

is executed until it is completed. Then the planner chooses another set of behaviors, and so on.

According to [89] the common components of hybrid architectures are:

1. ***Sequencer:*** this generates the set of behaviors to use in order to accomplish a task, and determines any sequences and activation conditions.
2. ***Resource Manager:*** this allocates resources to behaviors.
3. ***Cartographer:*** this is responsible for creating, storing and maintaining a map or special information, plus methods for accessing the data. It often contains a global world model and knowledge representation, even if it is not a map.
4. ***Mission Planner:*** this interacts with humans, operationalizes the commander in robot terms, and constructs a mission plan.
5. ***Performance Monitor and Problem Solving:*** this allows the robot to be aware of whether or not it is making progress.

There are three kinds of styles in hybrid architectures [89]:

1. *Managerial styles:* these are recognizable by their decomposition of responsibilities in a way similar to business management. The top level is in charge of performing high level planning; the plan is then passed to subordinates, who refine it and gather resources; it is then handed down to the lowest level workers, who behave reactively. The results at the lowest level can be seen at the higher levels, which in turn can issue new directives. A layer can modify the layer below it. Each layer attempts to carry out its directive, identify problems and correct them locally. Only when a layer cannot solve its own problem does it ask for help from a superior one (*fail upwards*). Architectures belonging to this style are AuRA [2], Socio-Intentional [64], DAMN [111], Yavuz [134] and Busquets [15] architectures, and Tripodal [63].
2. *State hierarchies:* these use the knowledge of the robot's state to distinguish between reactive and deliberative activities. Reactive behaviors are viewed as having no state or self-awareness, and function only in the present. Deliberative functions can be divided into those that require knowledge about the robot's past state (where it is in a sequence of commands) and about the future (mission and path planning). Architectures belonging to this style are 3-Tiered (3T) [10], ATLANTIS [44], SSS [24], ARA [92] and Berra [74].
3. *Model-oriented styles:* these have a more top-down, symbolic flavor than managerial or state-hierarchies. One hallmark of these architectures is that they concentrate

symbolic manipulation around a global world model. Unlike most other hybrid architectures, this global world model also serves to supply perception to the behaviors. In this case, the global world model serves as a virtual sensor. Architectures belonging to this style are Saphira [66] and Task Control Architecture (TCA) [117].

Next, several hybrid architectures are described, following the criteria described in the Introduction of this chapter. They are organized chronologically.

2.4.1 AUTONOMOUS ROBOT ARCHITECTURE - AURA (1987)

AuRA [2] is based on the schema theory, and consists of five major subsystems, equivalent to object-oriented classes. They are as follows (see Figure 2.7):

- *A cartographic subsystem* that encapsulates all the map making and reading functions needed for navigation. It contains a long-term memory (LTM, *a priori* knowledge) and a short-term memory (STM, containing the acquired perceptual model of the world).

- *A planning subsystem* that is responsible for mission and task planning. It has as its components the mission planner, the navigator (spatial reasoner in the most recent implementations of AuRA) and the pilot. The mission planner serves as the human interface; the navigator computes a path to achieve the goal and breaks it into subtasks. Finally, the pilot takes the subtasks and gets relevant information to generate behaviors.

- *A homeostatic control subsystem* that modifies the relationship between behaviors by changing their gains as a function of different constraints and in response to internal needs (self-consistency)

- *A perceptual subsystem* that gets sensory information through perceptual schemas. This subsystem extracts information from the environment, structures it in a coherent and consistent manner and delivers it to the cartographer and motor schema manager.

- *A motor subsystem* that contains the motor schemas that form the behavior schemas. This subsystem allows the robot to interact with the environment in response to sensory stimuli and plans. It contains the motor schema manager that is responsible for controlling and monitoring the behavioral process at run-time.

In AuRA several motor schemas can be selected to be active at the same time. To produce the steering and velocity commands that the robot will be provided with, a potential field methodology is used. This methodology creates velocity vectors as the vectorial sum of all the commands provided by the active motor schemas.

Actual implementations of AuRA have incorporated learning techniques such as on-line adaptation, case-based reasoning and genetic algorithms.

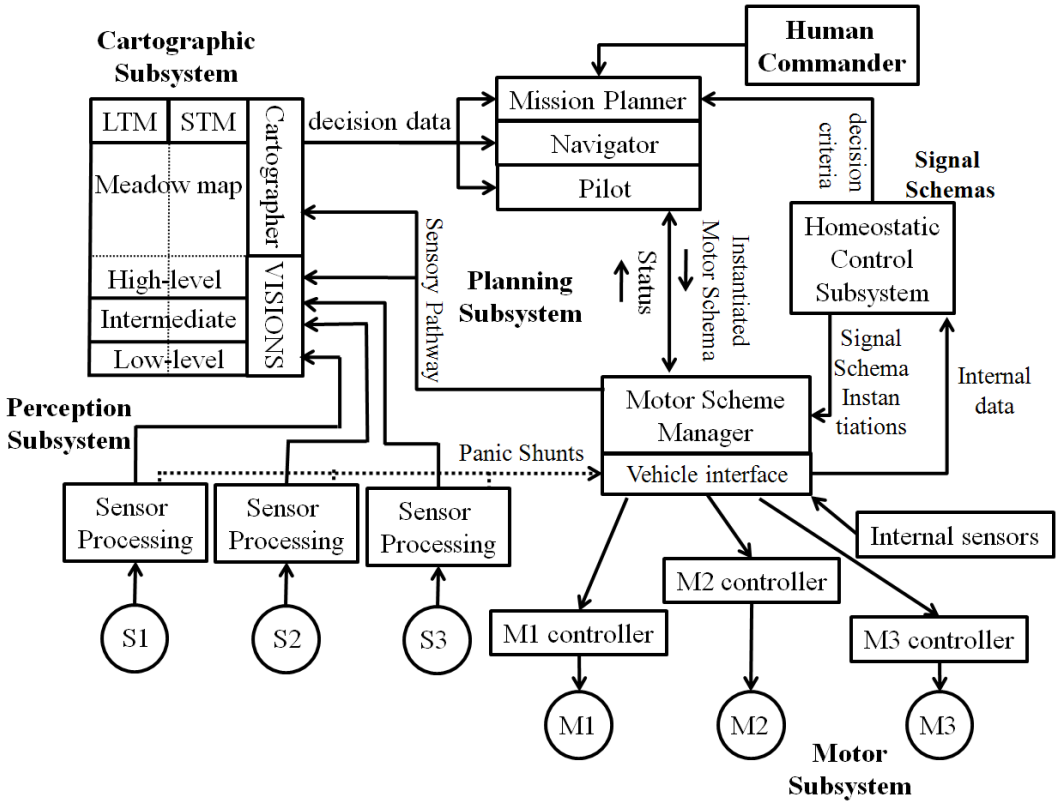


Figure 2.7: Layout of AuRa architecture (extracted from [2]).

2.4.2 A Three-Layer Architecture for Navigating Through Intricate Situations - ATLANTIS (1991)

ATLANTIS [44] is a heterogeneous and asynchronous architecture. It can handle multiple tasks in real time in noisy, partially unpredictable environments. None of the layers is in charge of the others, and activity is spread throughout the architecture. ATLANTIS smoothly integrates traditional symbolic planning into real-world embedded reactive systems. ATLANTIS consists of the controller (reactive control mechanism), the sequencer and the deliberator (see Figure 2.8). The reactive controller manages collections of primitive activities; the sequencer handles the initiation and termination of low-level activities and addresses reactive-system failures to complete the task; and the deliberator is in charge of planning and world modeling.

First, the control layer is programmed using ALFA (A Language For Action), a LISP-based program language. Programs in ALFA are composed of computational modules that are connected with each other and with the environment through communication channels. Second, in the sequencer, activities have a list of resources that they require and semaphores to prevent conflicts, so activities that interfere with each other are not enabled simultaneously (temporal sequencing methodology), similar to RAP [39]. In addition, if an activity is interrupted, the systems ensure that the activity is properly terminated (resources are released and the activity disabled). In 1994, monitor routines were introduced in order to determine *cognizant failures* and interrupt the system if one occurs, thereby providing an opportunity for plan restructuring. The notion of cognizant failure refers to the robot's ability to recognize on its own when it has not completed or cannot complete a task. Finally, the

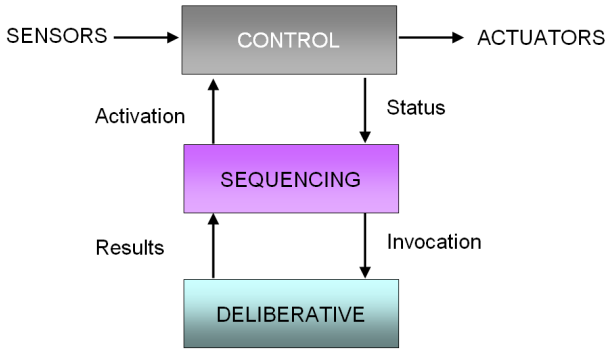


Figure 2.8: Layout of ATLANTIS architecture (extracted from [3]).

deliberator has typical AI algorithms implemented as a set of LISP programs and are used to plan and to maintain the world model. Algorithms are specific for the task at hand. Plans are stored in a database accessible by the sequencer, and they are used only as advice (not a decree) for the sequencer.

2.4.3 Servo, Subsumption and Symbolic Architecture - SSS (1992)

SSS [24] is a three-layered architecture that has Servo, Subsumption and Symbolic systems, as shown in Figure 2.9. In this work, the new ideas presented are an evolution of Brook's subsumption architecture. The robot's controller is composed of over 40 separate processes running in a loosely connected network of 24 processors. In this implementation there are a number of modules, each of them forming small parts of the overall behavior. Modules are complete, self-contained control systems which, based on perceptual information, generate the appropriate actuator's commands. Competing commands are coordinated using a priority-based arbitration mechanism; only one module at a time can gain control of the resource (actuator). There is no direct communication among modules, or a globally accessible short-term memory buffer. Communication can occur either from sensors to modules or from modules to the arbitration nodes. As in subsumption architecture, communication is achieved through ”wires”. Inhibition and suppression mechanisms among modules are still present in this proposal.

In SSS, the above ideas have been implemented in the layers of the architecture (see Figure 2.9). These layers implement different control techniques that are characterized by their treatment of space and time. The layers have special interfaces that allow them to cooperate effectively. The servo layer is continuous in space and time; the subsumption layer is discrete in space and continuous in time and the symbolic layer is discrete in time and space. The servo and the subsumption layers are equivalent to subsumption architecture [13]. The symbolic layer handles path planning and maintains a geometric map of the robot's world. Interfaces between layers have been carefully designed in order to use the three different technologies applied in each layer.

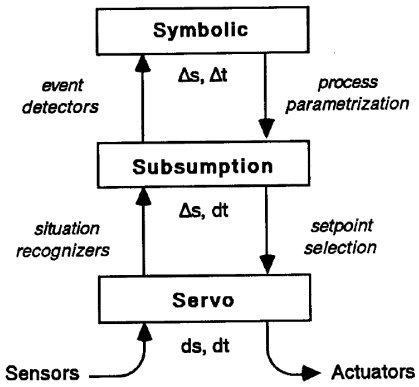


Figure 2.9: Layout of SSS architecture (extracted from [24]).

2.4.4 Task Control Architecture - TCA (1994)

TCA [117] has more of an operating system flavor than a general purpose architecture. There are no behaviors per se but many of the low level tasks resemble behaviors. It uses dedicated sensing structures which can be thought of as a distributed global world model. It has two layers, the deliberative and the reactive layers, as shown in Figure 2.10. The former is, at the same time, subdivided into three systems: task scheduling, which uses the PRODIGY planner [130], path planning and navigation, which uses a Partially Observable Markov Decision Process (POMDP) [59] to determine current actions and the present and past location of the robot. In the reactive layer the curvature-velocity method (CVM) [116] is used to respond to the motion of the robot with a smooth trajectory and obstacle avoidance. In this architecture there is a central unit, the central control, which registers all the modules that are in use and handles ethernet communication between modules. Modules can run on one or more computers.

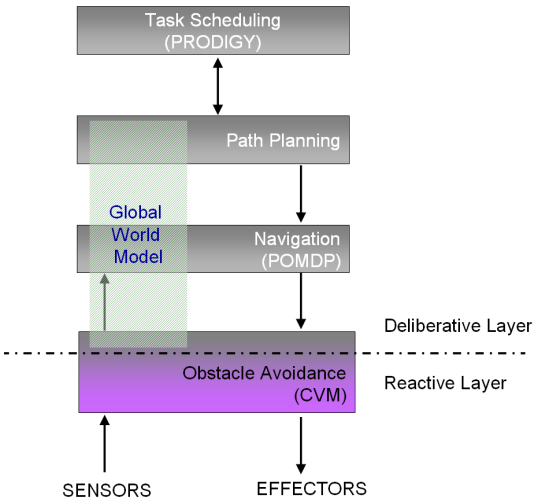


Figure 2.10: Layout of Task Control Architecture (extracted from [89]).

2.4.5 Distributed Architecture for Mobile Navigation - DAMN (1997)

In DAMN [111] architecture there is no hierarchy; instead multiple modules concurrently share the control of the robot. These modules communicate with the command arbitration module (a central arbiter) and vote for actions that satisfy its objectives and against those that do not, without regard for the level of planning involved (see Figure 2.11). The arbiter generates the appropriate control commands based on the combination of the modules' votes. There can be many independent arbiters (normally one for each degree of freedom) responsible for different aspects of control that can communicate if necessary. Behaviors operate asynchronously and concurrently, sending the outputs to the arbiter (or arbiters if appropriate) at a rate adequate for the particular function. Also, several arbiters can run in parallel and communicate among themselves for loose coordination while an arbiter can control multiple degrees of freedom when they are interdependent and must be jointly controlled.

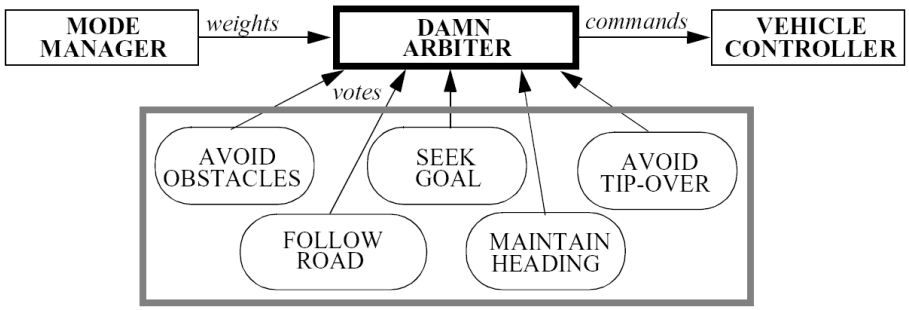


Figure 2.11: Layout of DAMN Architecture (extracted from [111]).

Four different arbitration methods have been distinguished, as follows:

- **Constraint arbitration scheme:** behaviors calculate the command action that maximizes the constraints they want to enforce and send these values to the arbiter, which selects the lowest of all of them, in this way satisfying all the constraints.

- **Actuation arbitration scheme:** behaviors vote for and against several control command alternatives. The arbiter counts these votes and selects the command that has the highest score.

- **Effect arbitration scheme:** similar to the actuation arbitration scheme except that the behaviors vote for and against several alternatives for the desired effect of the mechanisms being controlled (instead of for direct control of the actuator).

- **Map-based utility arbitration scheme:** behaviors express the utility of possible world states that can be represented within a map, and the arbiter fuses these utilities in order to determine which states are attainable and how to achieve them. In this case, utility fusion consists of selecting the state that has the greatest utility.

2.4.6 3T (1997)

3T [10] architecture is the merging of ATLANTIS architecture [44], the NaT (navigation templates) system [119] and the RAP system [39]. It has three different layers, one reactive, one deliberative and one which serves as an interface between the two. The three layers run on three different computers. The top layer of 3T is the Planner, whose functions are the mission planner and the cartographer. The middle layer is the Sequencer, which uses the RAPs planning technique to develop the task network specifying the sequence of execution for the behaviors. The bottom layer is the Controller or Skill Manager, which contains the set of possible skills.

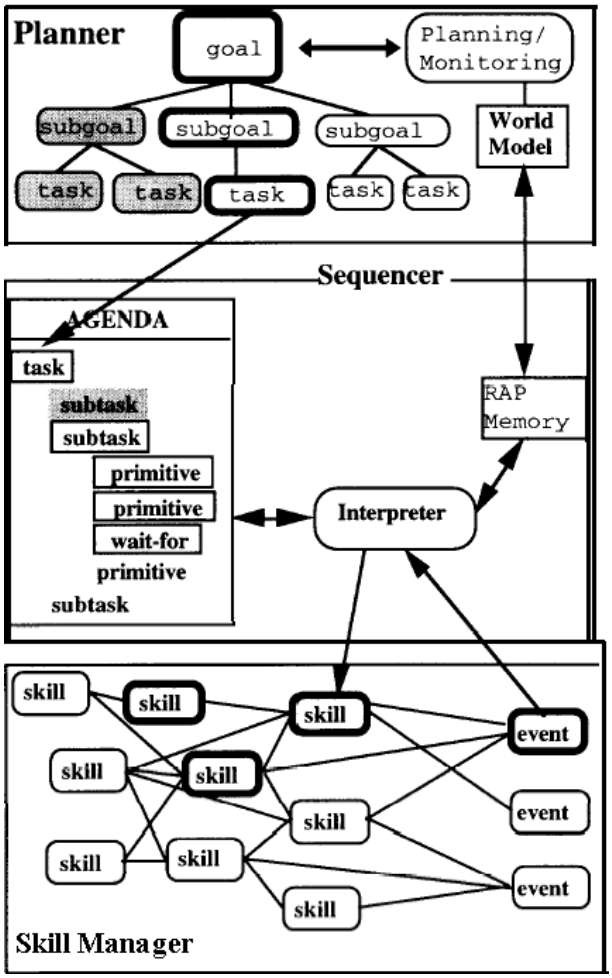


Figure 2.12: Layout of 3T Architecture (extracted from [11]).

2.4.7 BEHAVIOR-BASED ROBOT RESEARCH ARCHITECTURE - BERRA (2000)

BERRA [74] is a three-layered architecture, with a deliberative, a task execution and a reactive layer (see Figure 2.13). The deliberative layer has a human - robot interface (HRI) and a planner. The task execution layer has the task execution supervisor and the localizer. The reactive layer is formed by behaviors, resources and controllers. The HRI understands gesture and speech and also has a voice synthesizer for feedback. The planner decomposes commands from the HRI into states and state data. The states represent a certain configuration of the reactive layer. The task execution supervisor (TES) receives information from the planner and translates it into a configuration of reactive components. It also informs the controllers which behaviors should send data to them. The localizer is in charge of determining where the robot is and to track its position. Resources are for sharing sensor data. Clients of resources are behaviors or other resources. When a resource does not have a client it enters into the idle mode. In order to obtain information from a resource, clients must establish a connection and a kind of communication paradigm along with it (pull, push or synchronous). The behaviors are the coupling between sensors and actuators. They can be clients of more than one resource and accept connections from the controllers. They receive data from the TES and inform this module, if necessary, about the success or failure of an action. The controllers are in charge of sending direct commands to the actuators. The TES sends directives to the controllers to inform them about which behaviors they should get data from. Data coming from behaviors is arbitrated or fused in order to produce the control signal to be sent to the actuators. There is also a process manager that tracks all the processes used in the system.

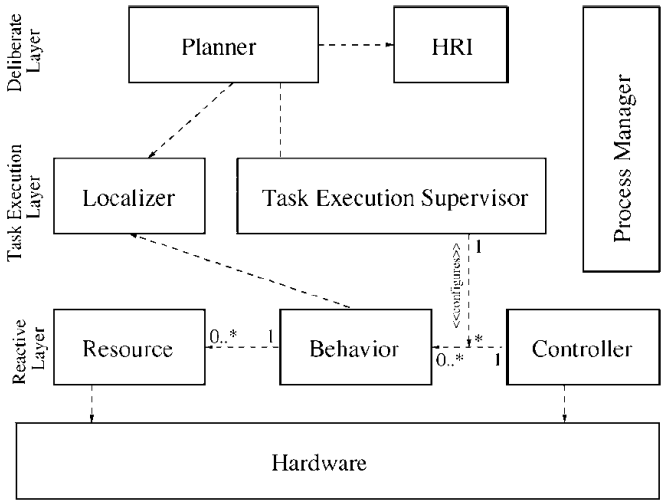


Figure 2.13: Layout of BERRA Architecture (extracted from [74]).

2.4.8 YAVUZ (2002)

The control system, presented in [134] is a modular hierarchical architecture with three layers, as can be seen in Figure 2.14. Deliberative actions are carried out in the upper layer, while behavior based decisions are performed in the lower layer. The deliberative layer selects the active behaviors depending on the operation mode, using a fuzzy behavior

analysis-integration and command arbitration module. The architecture has three kinds of operation modes: playback operation, teaching operation and manual operation. This last mode is intended for tele-operation while the other two methods are for self-supervised goal oriented autonomous operation. In this case, teaching must first be carried out by transferring task and goal information to the robot. Then, playback mode must be run to execute the tasks in an autonomous manner.

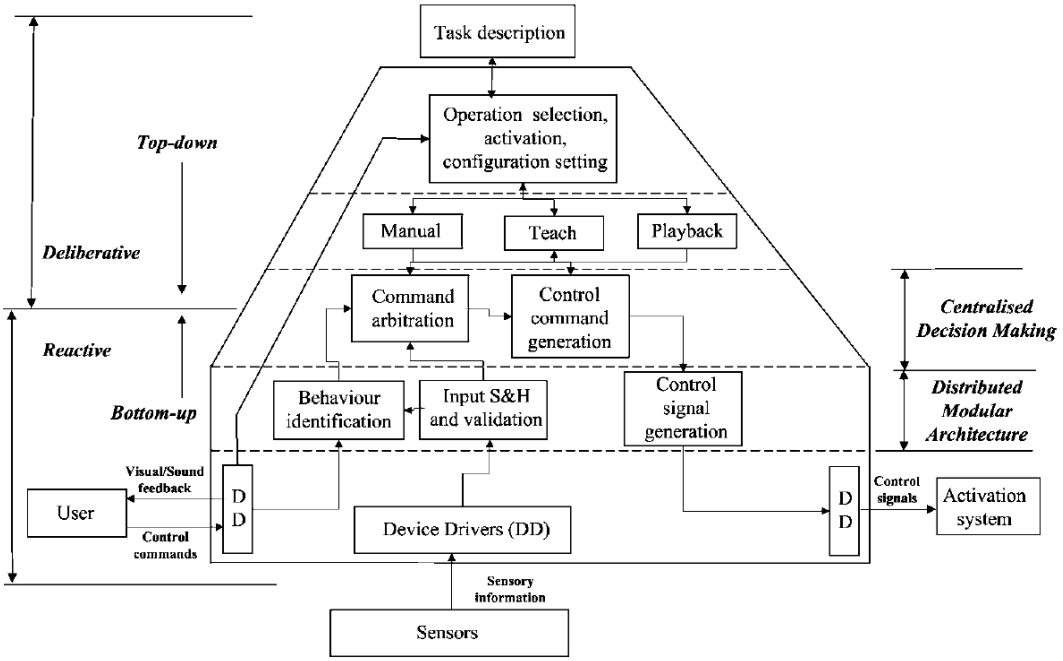


Figure 2.14: Layout of Yavuz architecture (extracted from [134]).

2.4.9 Tripodal Schematic Control Architecture (2006)

The architecture presented in [63] follows the three-layered model: the deliberative, the sequencing and the reactive layers (see Figure 2.15). The deliberative layer has two main tasks: the interface with the user and execution of the central planning process. The sequencing layer is formed by asynchronous and non-real-time components that are grouped into components related to supervision and low-level configuration tasks, and components that implement complex algorithms to extract advance information from raw sensor data. Finally, the reactive layer contains hardware-related and real-time components. It has a behavior coordinator that arbitrates control commands from one or more behaviors. Communication between components is performed in two ways: synchronous/asynchronous and pull/push relations. In synchronous messages the client component waits for a response from the provider component without performing any other task, whereas in asynchronous messages, the client component can continue working while the provider is processing the request. On the other hand, in the case of pull interaction the provider does not know the status of other components and does not provide any information unless requested, while in the case of push interaction, the provider actively informs as soon as possible when a change occurs.

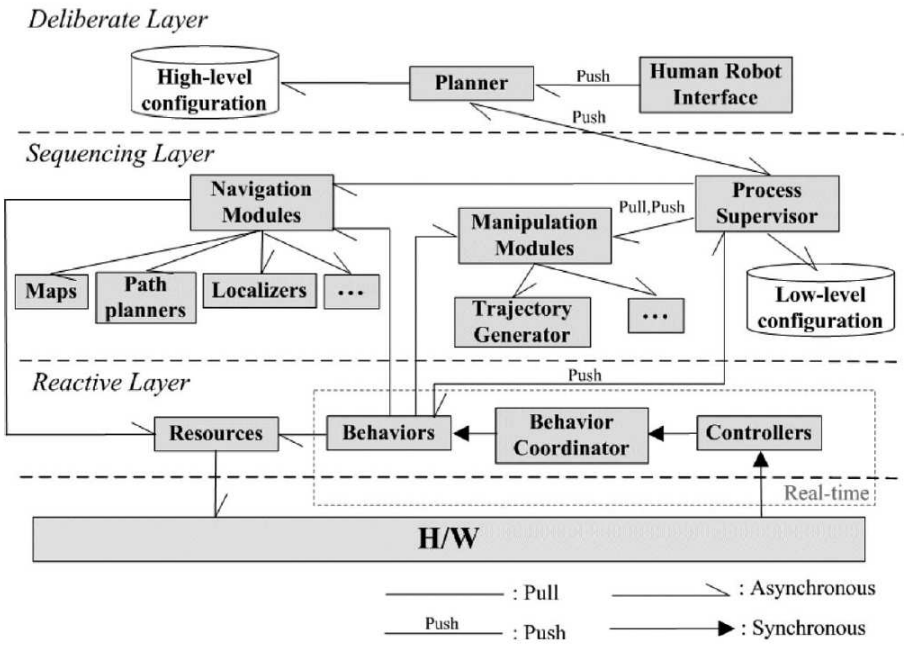


Figure 2.15: Layout of Tripodal Schematic Control Architecture (extracted from [63]).

2.5 Multi-agent Single Robot Approaches

Most of the above hybrid approaches implement the architecture as modules (i.e., object oriented). Recently, the agent paradigm has offered a new way of developing hybrid robust architectures, at a higher abstract level.

There are some examples of the use of multi-agent systems to control a single robot. In these architectures, several agents share control of a unique body. Moreover, the term agent is used with different meanings in the architectures listed above, according to the year the architecture was developed and the field where it comes from.

However, in each of the architectures, the following design decisions have to be tackled:

- What an agent is, which kind of deliberative / reactive behavior is in charge, how it is organized.
- Coordination: how agents coordinate in order to deploy the robot's actions.

2.5.1 Saphira (1997)

Saphira [66] architecture has two layers, deliberative and reactive (see Figure 2.16). The deliberative layer includes the planning agent, some deliberation activities divided into several different software agents, the local perceptual space agent, the topological planner agent, the navigation tasks agent and the localization and map maintenance agent. The reactive layer includes the reactive behaviors and the fuzzy module that fuses the competing demands from the behaviors (i.e. coordinates the behaviors).

The core of the architecture is the planning agent. It is based on the Procedural Reason-

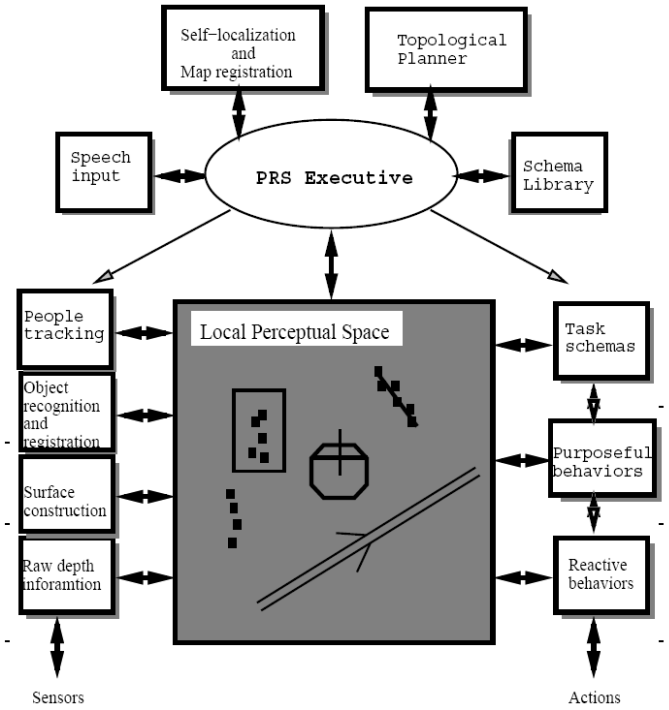


Figure 2.16: Layout of Saphira Architecture (extracted from [66]).

ing System-lite (PRS-Lite) [65] reactive planner. It is capable of taking natural language voice commands and transforming them into navigation tasks and perceptual recognition routines. Both planning and execution relies on the Local Perceptual Space agent, the central world model. Coordination of behavior is done using fuzzy logics. The output of each behavior is a *Desirability function*, i.e, a measure of how much each possible control command is desirable given the current state. The control action to be sent to the actuators is the weighted average of the desirability functions of the activated behaviors. Activation/deactivation of behaviors is the main purpose of the PRS-lite system. This system is represented as activity schemas, a parameterized state machines whose arcs are labeled with goals to be achieved. Each schema embodies procedural knowledge of how to attain an objective via a sequence of subgoals, perceptual checks, primitive actions and behaviors. Activity schemas are launched by instantiating their parameters and *intending* (instantiated schemas are referred to as *intentions*) them into the system.

2.5.2 Autonomous Mobile Robot Agent Architecture - ARA (1997)

ARA [92] is composed of a mobile robot platform, a control system and an interface module, as depicted in 2.17.

The control system is ARCoS (Autonomous Robot Control System). It can be considered as a society of agents from whose coexistence and cooperation emerge the functionality of the mobile robot. These agents are organized in three main communities: reflexive, reactive and cognitive. The first is responsible for the most basic abilities (e.g. survival instincts), the second is in charge of the most relevant abilities of the mobile robot, and the third decides the best actions to perform based on reliable knowledge about the environment. Reflexive agents run on the onboard computer, while reactive and cognitive agents run on different

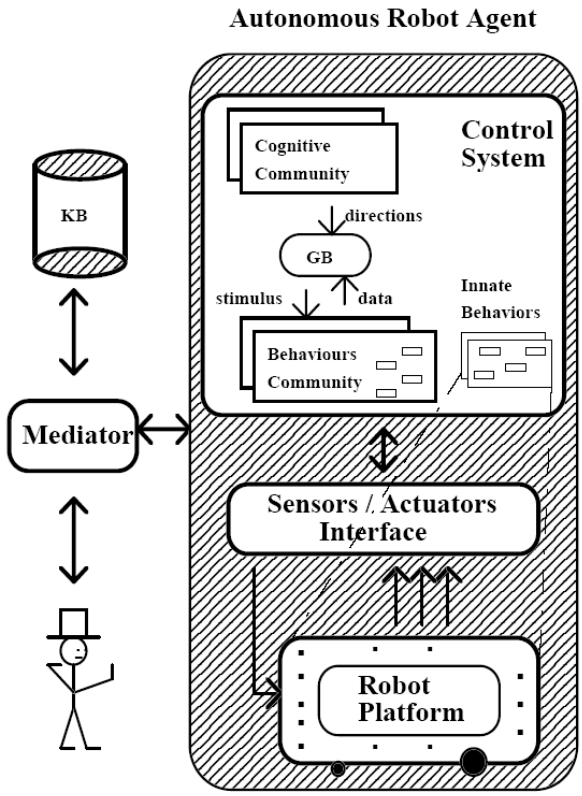


Figure 2.17: Layout of ARA Architecture (extracted from [93]).

machines. All of them communicates via sockets. To the Reflexive Community belong the agents that act in a pure stimulus-response way. To the Reactive Community belong the more elaborate reactive agents that have tight coupling between perception and action without heavy intermediate computation. In this community, the agents responsible for elementary abilities (they act together with the reflexive agents) are called Elementary Agents (EAs), and the agents responsible for the integration and coordination of several EAs in order to achieve a more complex competence are called High-Level Agents (HAs). The cognitive agents have access to topological and geographical information in the Knowledge Base according to the current task; they develop alternative abstract plans without details. The Green Board (GB) Agent acts as an interface agent between the Behavior and Cognitive Communities. It has the ability to convert the directives from the cognitive agents into specific stimulus to convey to the relevant behaviors.

In this architecture, if there are no appropriate actions to perform in a given situation, the Tutor Module takes control and proposes the most appropriate action, either by generating a random action to be executed, or using some specific built-in knowledge, or by means of human help.

In the reactive community, the HA agents perform high-level behaviors, through coordination of the EA agents. In ARA, four types of coordination mechanisms, in the form of operators, are implemented as stated in [93]:

- $+$ **operator:** specifies a combination of mutually exclusive behaviors. Due to mutually exclusive behaviors being activated by mutually exclusive conditions, when a behavior is activated it takes full control of the robot.

- $>$ **operator:** indicates a weighed combination of behaviors resulting in the output proposed by the higher activated behavior. The value of activation function, called activation level, is considered the factor that scores the behaviors. When a high-level behavior is defined through this operator, a priority factor must be assigned to each of the component behaviors.

- $\&$ **operator:** means a combination of multiple concurrently active behaviors. The output satisfies all the component behaviors because it is the intersection of every proposed output.

- $\sharp$ **operator:** suggests a combination of behaviors characterized by activation of one of the component behaviors. On the basis of a set of trigger conditions, just one of the behaviors can be activated at a time. Trigger conditions are predicates on data that characterize the current situation. The strategy implemented by this operator is a switching between behaviors, thereby assuring a coherent sequence of outputs.

2.5.3 BUSQUETS (2003)

This non-hierarchical architecture [15] is composed of three systems: the Pilot system, responsible for the motion of the robot, the Vision system, in charge of identifying and tracking landmarks, and the Navigation system, which chooses higher-level decisions in order to move the robot to a specified target (see Figure 2.18). Each system competes for the two available resources: motion control and camera control. But, also, each system must cooperate with the others in order to achieve the goal, a target position. The Navigation system is a multi-agent system composed of behavioral agents (each agent represent a specific behavior) and a coordinator agent.

Agents in this architecture can be classified into an executive system and a deliberative system. The former has access to the sensors and actuators of the robot, providing information gathered from sensors and offering services for using the actuators. The latter takes high level decisions and requires the services offered by the executive system.

In order to coordinate agents in the different systems, and in the Navigation system, a simple bidding mechanism is used. The agents bid according to the expected reward that they would receive if they were given control; they seek to maximize the overall system reward. Each system (executive and deliberative) generates bids for the services offered by executive systems, according to the internal expected utility associated with the provisioning of such services. A coordinator (block ”C” in Figure 2.18) receives these bids and decides which service has to engage each of the executive systems. The bid represents the urgency for a system of having a service engaged. The bids are in the range [0,1], with high bids meaning that the system really thinks that the service is the most appropriate at that moment, and low bids meaning that there is no urgency to having the service engaged.

This coordination is also carried out in the Navigation system, each behavioral agents send bids to the coordinator agent, and this agent determines the winning bid (which is used to coordinate with the other systems).

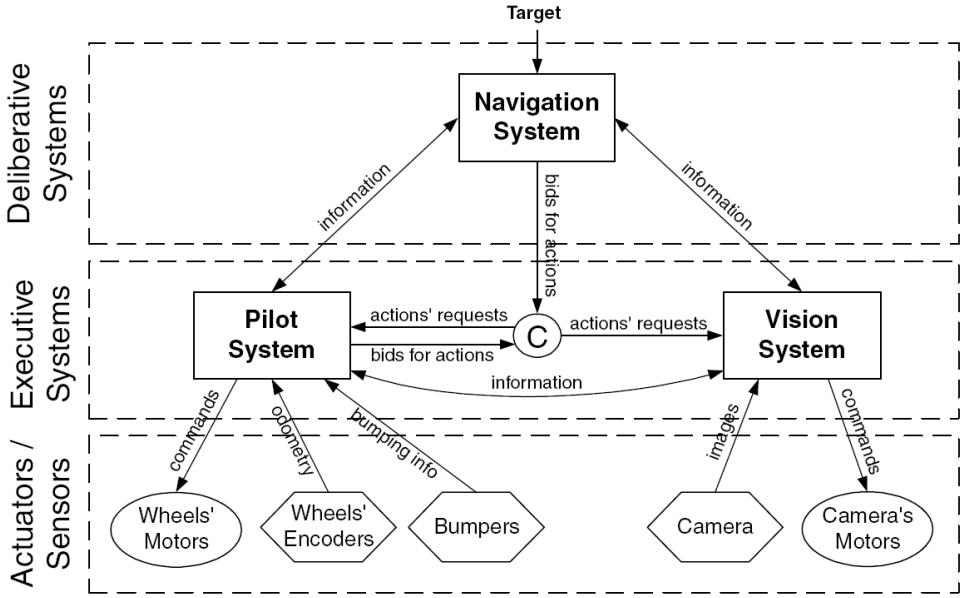


Figure 2.18: Layout of Busquets architecture (extracted from [15]).

2.5.4 Socio-Intentional Architectures (2006)

In [64] there is a proposal to develop multi-agent systems to control a mobile robot, using organizational styles. These styles are based on organization theory and strategic alliances. The former define organizational structures, which are used to understand the structure and design of an organization, while the latter models the strategic collaboration of independent organizational stakeholders who have agreed to pursue a set of shared business objectives. Both disciplines aim to identify and study organizational patterns that describe a system at a macroscopic level in terms of a manageable number of subsystems, components and modules inter-related through dependencies.

In spite of the fact that this theory can be applied to any MAS, in [64] two multi-agent architectures for mobile robot control are presented, following the *structure-in-5*, shown in Figure 2.19 and the *joint-venture* organizational models, as can be seen in Figure 2.20. In the former, five subsystems are placed following a start structure, where the navigator is in the middle as the central intermediate agent coordinating the movements of the robot to take responsibility for reliability tolerance and adaptability management. In the latter, the robot architecture is organized around a joint manager taking responsibility for coordination.

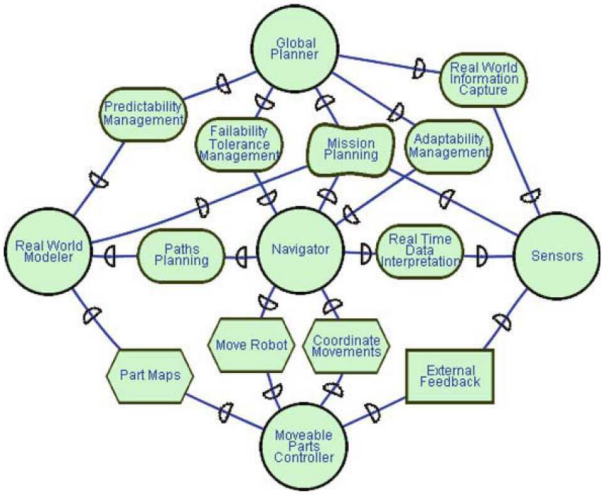


Figure 2.19: Layout of Structure-in-5 socio-intentional architecture (extracted from [64]).

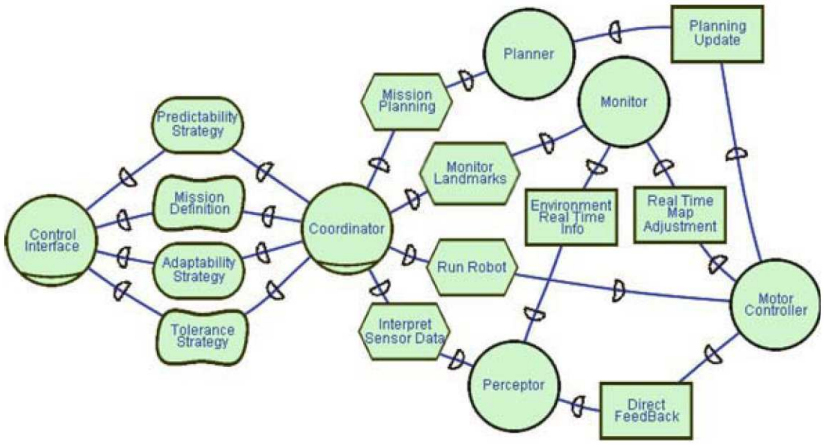


Figure 2.20: Layout of Joint-Venture socio-intentional architecture (extracted from [64]).

2.5.5 Virtual Operator Multi-Agent System (VOMAS) (2007)

VOMAS architecture [53] has been developed for a mobile robot capable of dynamic task switching in order to support spontaneous generation of a task without preplanning. There are three main agents: the virtual operator (VO), the robot agent (RA) and the user agent (UA). The VO agent deals with high-level planning to represent a task. The RA is a stationary agent in the robot context, and performs the functions that are highly dependent on the physical platform. The UA is the agent through which the user can control the robot. This architecture is depicted in Figure 2.21. The VO agents are designed to implement deliberative control and hardware-independent functions while the RA implements reactive control and hardware-dependent functions.

In order to assign a task to the robot, the user utilizes the UA to create the corresponding VO (the one specializing in the desired task). This VO can move from the user context to the robot context. After this migration, the VO and the RA cooperate and execute the task. In a situation where the robot needs to change the task, it can replace one VO with another.

The VO and the RA agents use an agent communication language (ACL) to coordinate between themselves. The defined protocols are the VOMAS-Robot-Request protocol to request control and the VOMAS-Robot-Leave-Request protocol to end the control. The RA agent uses a behavior-based architecture to implement the reactive behaviors, namely Obstacle Avoidance, Robot Avoidance, Goal Seeking, Road Following and Predecessor Following.

This architecture has been applied to a wheelchair robot and to a formation control (multi-robot system). In the former, the RA is designed based on DAMN [111]. The behaviors designed in the RA are obstacle avoidance, road following, and goal seeking. In the latter the RA is based on motor-schema, also a behavior-based architecture. The behaviors in the RA included obstacle avoidance, robot avoidance, goal seeking, and predecessor following.

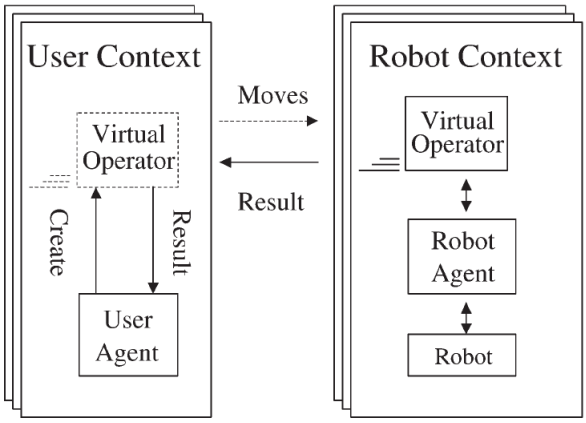


Figure 2.21: Layout of VOMAS architecture (extracted from [53]).

2.6 Multi-agent Multi-Robot Approaches

Multi-robot teams are desirable for exploration in hazardous environments (underwater, on distant planets, inside damaged structures, etc.); tasks beyond the limits of a single robot (cooperative pushing, assembly of large and complex structures, etc.); tasks that can be completed more rapidly by multiple robots (collecting trash, searching for land-mines, etc.); complex tasks where specialized and simpler vehicles may be less expensive than a multipurpose robot (exploration, communication, object retrieval on a planetary surface, etc.); and highly distributed sensing (hazardous chemical sensing, detection of explosives, etc.), motion sensing (detection of small earthquakes, movements of large number of people, etc.) or temperature sensing (detecting changes in water temperature in the ocean) [7].

Many cheap robots can cover more area than a single expensive robot, making multi-agents more cost effective. Building and using multi-robot teams can be easier, cheaper, more flexible and more fault-tolerant than having only one expensive and powerful robot.

Formally, a collection of two or more mobile robots working together are termed teams or *societies* of multiple mobile robots [89]. Multi-robot approaches are intended to exhibit cooperative behavior.

Thus multi-agent, multi-robot approaches study how to achieve cooperation by using the multi-agent paradigm. It should be observed then, that in this field, a robot is an agent (also called a physical agent). So the multi-agent paradigm is used **outside** the robot

architecture, the converse of approaches studied in the previous section, in which multi-agent architectures are used **inside** the robot. Even though objectives in the field of multi-agent multi-robot approaches are not the same than in multi-agent single robot approaches, there are many common aspects, such as coordination and communication, that justify the following overview.

When designing a multi-robot system various aspects must be considered [17],[3], [89], [7]: the group architecture, whether the system's control is centralized or decentralized; the differentiation of the agents, whether robots are homogeneous or heterogeneous; the ability of a given robot to recognize and model other robots; the communication structure and cooperation motivation.

First, the group architecture, upon which collective behaviors are implemented, determines the capabilities and limitations of the system. Group architecture can be centralized or decentralized. In a centralized architecture a single agent gives orders to other agents and supervises the correct performance of a cooperative task. This kind of architecture can be highly efficient (and inexpensive) when it comes to controlling a small group of robots, since there is a *supervisory robot* that decides which task to perform and *workers* that perform the given tasks through actuators. An example of centralized architectures is [62]. In this group architecture a computer with a sophisticated vision system senses all the robots' positions (the robots have an appropriate identification), performs the task allocation and controls each robot. However, the control of each member of the system (also termed colony) by an external controller is difficult or impossible as the colony grows in size [7].

On the other hand, decentralized architectures, inspired by insect societies, do not have an agent that controls the whole system and are a generalization of behavior-based control over multiple robots [7]. Decentralized architectures can be fully distributed, in which all agents are equal with respect to control, or hierarchical, in which they are locally centralized. Examples of decentralized architectures are [42], [101], [4] and [36].

Second, in multi-robot systems, robots can be homogeneous if the capabilities of individual robots are identical, and heterogeneous if not. Heterogeneity introduces complexity since task allocation becomes more difficult, and agents have a greater need to model other individuals in the group. In homogenous teams, agents may need to take on distinct roles that are either known at the design time or arise dynamically at run-time. Examples of homogeneous teams are [42] and [36] while [101] and [4] are heterogeneous teams.

Third, regarding the ability of a given robot to recognize and model other robots, if an agent can model the intentions, beliefs, actions, capabilities and states of other agents, this can lead to more effective cooperation between robots. [101] and [4] are architectures where there is a model of the agent itself, other agents and the environment.

Fourth, the communication structure is the last aspect in the group architecture. This parameter determines the possible modes of inter-agent interactions. The three major types of interactions are: via the environment, via sensing and via communication. Interactions via the environment are the simplest and the most limited. There is no explicit communication or interaction among agents; the environment itself is the communication medium. Interactions via sensing (arms-length relationships) refer to local interactions that occur between agents as a result of agents sensing each other, but without explicit communication. This type of interaction requires the ability of agents to distinguish between other agents in the group and other objects in the environment. Interactions via communication involve explicit communication with other agents, by direct (point-to-point) or broadcast intentional messages.

Finally, another important aspect in multi-robot system is the way cooperation is motivated and achieved. Cooperation can be explicitly designed into the system or can arise as a

result of the local interactions of a large number of homogenous robots, each of which has fairly limited capabilities on its own. So, multi-robot systems can be categorized into two groups: emergent social behavior or intentional cooperative behavior.

Emergent social behavior (named *eusocial behavior* in [17] and *swarm-type cooperation* in [101]) is found in many insect species, where individual agents are not very capable and a ”cooperative” behavior is necessary for the survival of the individuals in the colonies. ”Cooperative” behavior is motivated by their innate behavior and a seemingly intelligent behavior arises out of their interactions. This approach is useful for non-time critical applications involving numerous repetitions of the same activity over a relatively large area. Examples of swarm-type cooperation are found in [42] and [36].

On the other hand, with intentional cooperative behavior (also named *cooperative behavior* in [17] and *intentional cooperation* in [101]), cooperation is the result of interactions among a limited number of selfish heterogeneous agents. Cooperation is motivated by an intentional desire to cooperate in order to maximize individual utility; robots often have to deal with some sort of efficiency constraint. In this type of cooperation a proper mapping of subtasks to robots is dependent upon the capabilities and performance of each robot team member. This additional constraint brings many complications to a workable architecture for robot cooperation, and must be addressed explicitly to achieve the desired level of cooperation. This type of cooperation is found in [101].

Next, the most important multi-robot systems are given in order to exemplify the properties of these kinds of system. They are ordered according to the year they appeared.

2.6.1 ACTRESS (1989)

The ACTRESS (ACTor-based Robot and Equipments Synthetic System) [4] is based on the Universal Modular ACTOR Formalism. This formalism provides a computational model of information processing where data structures and control structures are represented by a single kind of object, called ”actors”, and a message passing between the ”actors”. In ACTRESS, actors are presented by the ”robotors” (robotic actors) that manage data and control the physical body. Robotors have the ability to make decisions (are autonomous) and to communicate with other robotors. This architecture is distributed and robotors are heterogeneous. They are interconnected by a communication network and communication protocols, at different abstraction levels, provide the means for group cast and negotiation mechanisms [82]. Robots in this architecture have a CPU, position and collision sensors, and a communication device.

2.6.2 CEBOT (1990)

CEBOT [42] (CEllular roBOTics system) is a decentralized, hierarchical architecture inspired by the cellular organization of biological entities that dynamically reconfigure their structure to an optimal configuration in response to changing environments. CEBOT units are autonomous and mobile robots that can be physically coupled to other robots, some of them being ”master robots”. In CEBOT there is hierarchy where master robots coordinate subtasks and communicate with other master robots in order to self-configure. This self-configuration is determined using a genetic algorithm. Robots in CEBOT are homogenous, equipped with collision and position sensors, a CPU and communication devices.

2.6.3 Behavior-based Cooperative Behavior (1992)

Mataric expanded the subsumption architecture to multi-robot systems [80, 81], in the ”Nerd Herd” multi-robot system. The ”Nerd Herd” is a group of 20 autonomous and homogeneous robots operating under fully distributed control, in which social rules determine overall team performance. Three kinds of experiments were carried out to test social rules: ignorant coexistence, informed coexistence and intelligent coexistence. In ignorant coexistence the robots in the team did not have any knowledge about other members in the team; other robots were treated as obstacles. When performing the mission, robots spent most of the time avoiding each other, and the team as a whole made slow progress in completing the mission. The experiment also proved that more time was needed to complete the mission when more robots were in the team. In informed coexistence, robots recognized team members and their interactions were governed by a simple social rule. When a robot detected another robot it stopped moving and waited some time before continuing with its mission. Team progress was improved and equivalent to a mission carried out 20 times by the same robot. Finally, in intelligent coexistence, robots could recognize team members and were provided with a mechanism for handling congestion and collisions. As a result, the robots exhibited flocking behavior and completed their mission faster and reduced traffic hold-ups. Mataric's work proved that simple social rules can reduce interference among robots on the same team.

2.6.4 ALLIANCE (1998)

ALLIANCE [101] architecture is a fault-tolerant, adaptive, distributed behavior-based multi-robot control system, that can perform (in dynamic environments) missions composed of independent tasks that may have ordering dependencies. Robots are heterogeneous, fully autonomous and can perform useful action even when other robots fail. Robots are able, with some finite probability, to sense the effects of their own actions and to determine those of other members of the team through perception and via explicit broadcast communication. This allows the robot to consider the environment, its own internal state and the actions of other robots, in order to select the appropriate actions throughout missions. To achieve adaptive action selection, a mechanism based upon the use of mathematically-modeled motivations within each robot, such as impatience and acquiescence, is used. Impatience motivation enables a robot to handle situations when other robots (outside it) fail in performing a given task. Acquiescence motivation enables a robot to handle situations in which it, itself, fails to properly perform its task.

These motivational behaviors activate ”the behavior sets” that are either active as a group or are hibernating. Each behavior set of a robot corresponds to those levels of competence required to perform some high-level task achieving function.

An extension to ALLIANCE, is L-ALLIANCE (Learning ALLIANCE) [100] that uses reinforcement learning to adjust the parameters controlling behavior set activation.

2.6.5 SWARM-bots (2002)

Swarm robotics consists of the application of swarm intelligence to the control of robotic swarms, emphasizing decentralization of control, limited-communication abilities among robots, use of local information, emergence of global behavior and robustness [36].

Swarm-bots is an artifact composed of a swarm of s-bots, mobile robots with the ability to self-organize and reconfigure according to environmental changes. Each s-bot is a simple but fully autonomous unit capable of displacement, sensing and acting based on local information and decisions [84]. In this architecture artificial evolution has been used for synthesizing the controllers for the s-bots and for obtaining self-organization in the robotic system [36], proving that evolution is able to produce a self-organized system that relies on simple and general rules, that can adapt to environmental changes.

2.7 Robotic Development Environments

Robotic development environments (RDE) have been developed to facilitate research into autonomous robotics and to help architecture designers manage the complexity of agents [68]. RDEs are frameworks that are aimed at developing reusable, portable, modular, flexible and extendable software that can support heterogeneous robotic platforms.

In [68] there is a comparison of nine RDEs in terms of *specification*, which includes formalisms, methodologies, and design tools; *platform support*, which is related to the hardware and its low-level interface (e.g., the operating system); *infrastructure*, which refers to components and capabilities that are part of the RDE, but not the ”agent architecture proper”; and *implementation*, which includes aspects of application development (including predefined components used in an agent architecture). In this work, the RDEs have been compared according to three types of score in order to provide designers with useful information when choosing an RDE. A summary of the analyzed RDEs can be found at the end of this section.

Next there is a list of other mobile robot architectures that can be considered as RDEs and that are not analyzed in [68]. They have been used either to implement a real robot control, as in [12], or to model some existing architecture as in [90], in a modular/multi-agent way. Moreover, there are two wide-ranging projects, CLARAty [91] and OROCOS [14], whose aim is to become general-purpose robot control software packages.

2.7.1 ARTIS (1999)

ARTIS architecture [12] is meant to develop agents that need to work in hard real-time[1] environments. The architecture guarantees the execution of the entire system specification by means of an off-line analysis of the specification. However, it does not force task sequence execution. An Artis Agent (AA) decides the next task to be executed at runtime, allowing it to adapt itself to changes in the environment. The AA reasoning process can be divided into two stages. The first is a mandatory time-bound phase. It obtains an initial result of satisfactory quality. After that, if there is time available, the AA can use this time for second reasoning stage. This is an optional stage and does not guarantee a response. It usually produces a higher quality result through intelligent, utility-based, problem-solving methods.

The architecture of an Artis Agent can be viewed from two different perspectives: the user model (high-level model) and the system model (low-level model). The user model offers the developers view of the architecture, while the system model is the execution framework used to construct the final version of the agent. From the user model point of view, the AA architecture is an extension of the blackboard model, which is adapted to work in hard, real-time environments. From the system model, there are two levels of agents: the so-called ARTIS Agent (AA) and a set of agents (in-agent -Internal Agent-) within it.

[1] Hard real-time means that the execution of a task after its deadline is completely useless.

The ARTIS Agent is depicted in Figure 2.22. It is made up of:

- A set of sensors and effectors so that it can interact with the environment (due to environment features, the perception and action processes are time-bounded).
- A control module that is responsible for the real-time execution of each component that belongs to the AA.
- A reflex level that assures an answer to any critical environment event.
- An Intelligent Server (IS) which is used for higher quality answers if there is enough time (calculated on the basis of the cognitive levels of the different in-agents), and which handles non-critical sporadic environment events. It is therefore, in charge of all the non-critical cognition of the AA.

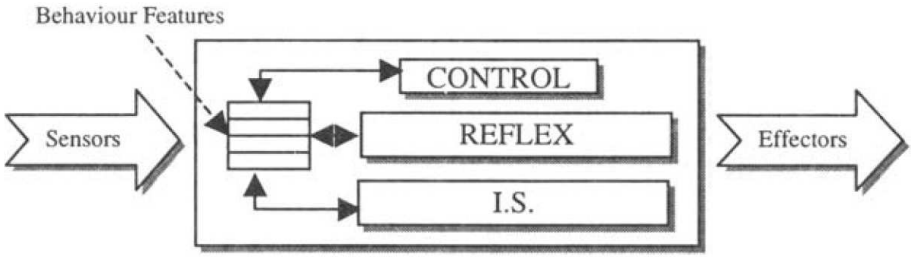


Figure 2.22: Layout of ARTIS Architecture (extracted from [12]).

As an example, ARTIS has been applied to control a mobile robot. From the user model, there is only one ARTIS agent that controls the robot, but from the system model, each behavior of the robot is implemented as an in-agent of the ARTIS Agent. The in-agents present in the AA are: avoid obstacles, malfunction monitoring, localization, trajectory planner, job planner and radio communication agents. The avoid obstacle in-agent has the highest priority so it modifies the action commands sent by the trajectory planner when needed.

ARTIS is the base of the SIMBA [58] platform, which allows the development of real-time multi-agent systems (RTMAS). The SIMBA platform (see Figure 2.23) consists of a set of ARTIS agents and a special agent (Manager Platform Agent MPA) which controls the services specified in the standard FIPA [38]. These services are: agent management services (also called white pages service AMS) and directory management services (also called yellow pages DF). This agent also controls inter-operability with other FIPA platforms across an agent communication channel (ACC). With this platform, the ARTIS agents are transformed into social real-time agents that can communicate with other agents by means of an agent communication language (ACL).

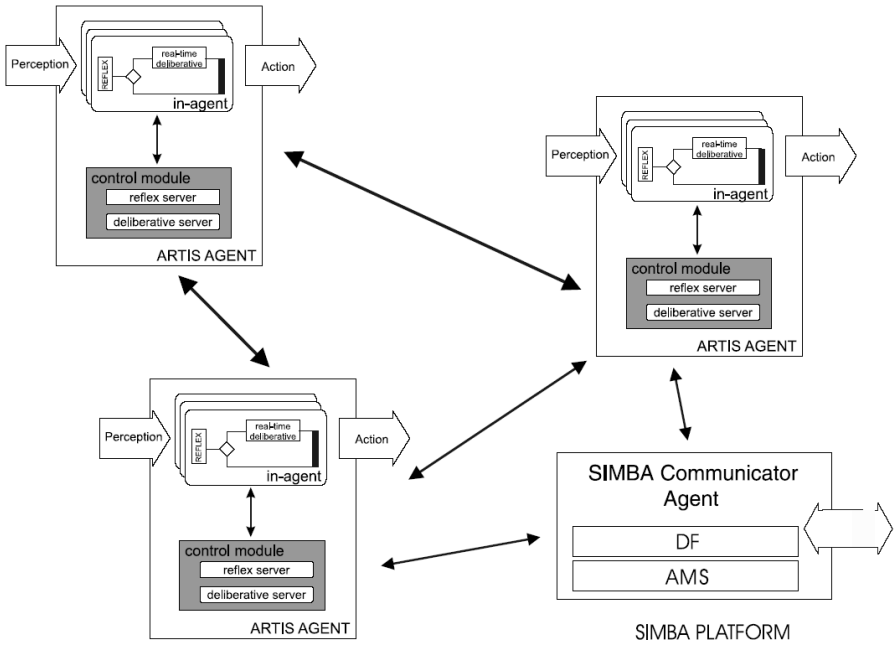


Figure 2.23: Layout of the SIMBA Platform (extracted from [18]).

2.7.2 Open Robot Control Software - OROCOS (2001)

OROCOS [14] is a project that aims to develop a general-purpose, free software, modular framework for robot and machine control. At present, the OROCOS project supports four C++ libraries: the Real-Time Toolkit, the Kinematics and Dynamics Library, the Bayesian Filtering Library and the Orocos Component Library, as depicted in Figure 2.24.

The Real-Time Toolkit provides a context in which to implement real-time and non real-time control systems. It is a library that allows application designers to build highly configurable and interactive component-based real-time control applications. It also, offers abstract interfaces to common hardware such as encoders, AD/DA conversion, etc., as well as (links to) a number of relevant hardware device drivers.

The Kinematics and Dynamics Library provides a framework for modeling and the computation of kinematic chains. It provides a library for geometrical objects, kinematic chains of various families, and their motion specification and interpolation.

The Bayesian Filtering Library provides a framework for recursive information processing algorithm and estimation algorithm based on Bayes' rule.

The Orocos Components Library provides some ready to use control components.

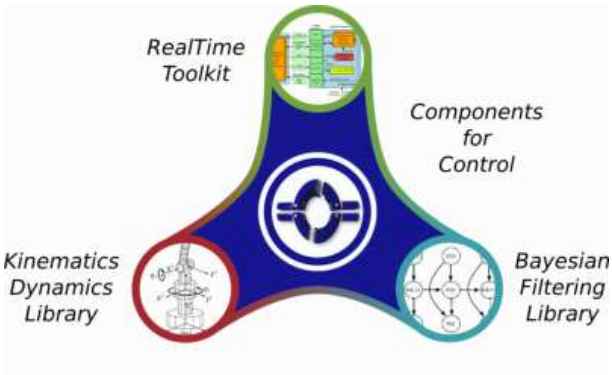


Figure 2.24: Layout of OROCO (extracted from [98]).

2.7.3 INTELLIGENT DISTRIBUTED EXECUTION ARCHITECTURE - IDEA (2002)

IDEA [90] is a multi-agent platform that provides a unified representational and computational framework for planning and execution for each agent. The main components of IDEA, as depicted in Figure 2.25, are:

Tokens and procedures: a token is the fundamental unit of execution. A token is a time interval during which the agent executes a procedure. A procedure is a tuple of input, mode and output arguments and a status. A procedure can be executed when all the input arguments are known; the time when the procedure starts is the token start time. At any time during the execution the procedure can return a value for each output. The execution of a procedure is terminated when a status value is returned or the agent interrupts the token's execution.

Virtual machine: which integrates planning as the reasoning module at the core of the execution engine. The virtual machine is composed of four main components whose interplay provides the basis for the agents autonomous behavior: the domain model, the plan database, the plan runner, and the reactive planner. The Domain Model describes which procedures can be exchanged with which external agents. It also specifies which procedure arguments are expected to be determined before a goal is sent to another agent (input arguments) and on which arguments the agent is expecting execution feedback from some other agent executing the token (output and status arguments). The Plan Database describes the portion of the past that is remembered, the tokens currently in execution, and the currently known future tokens, including all the possible ways in which they can execute. Each token parameter (input, mode, output, status, and start and end time) has an associated variable. All these variables are connected by explicit constraints into a single constraint network. The Plan Runner is the core execution component of the agent. It is activated asynchronously when either a message has been received from another agent or an internal timer has gone off. The Reactive Planner is in charge of returning a plan that is locally executable. It must guarantee the consistency of token parameters in terms of the plan's constraints and support for the token according to the domain model.

Communication wrapper: which defines a simple communication protocol for interaction among separate IDEA agents; an underlying inter-process communication mechanism. It is in charge of sending messages that initiate the execution of procedures or receiving goals that are treated by the agent as tokens. The format of the allowable communications is governed by the domain model.

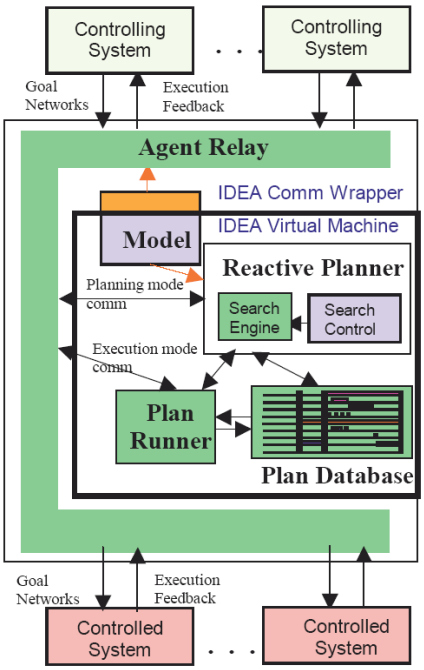


Figure 2.25: Layout of IDEA architecture (extracted from [90]).

This multi-agent platform has been used to implement the Remote Agent (RA) that controls some of NASA's space vehicles.

2.7.4 Remote Objects Control Interface - ROCI (2003)

ROCI [20] is a self-describing, object oriented, strongly typed programming framework for distributed sensors and actuators that facilitates the development of robust applications for dynamic multi-robot teams and the implementation of tasks in a single robot.

The basic units of ROCI are self-contained, reusable modules that encapsulate processes that generate outputs acting on input data (see Figure 2.26). The interconnection of these modules can build complex tasks. Communication among modules is designed to be network transparent and if connections are not within the same tasks domains, ROCI creates a Remote Procedure Call channel.

ROCI has a kernel that must be running on every entity that forms part of the ROCI network. This kernel manages the Remote Procedure Call (RPC) system, the real-time network database, module and task allocation and injection, and a Web Services like interface for remote monitoring and control. The RPC system provides interfaces for module management, injection and communication, as well as a web-based interface to the current status of the network. The real time database contains information on all of the modules, tasks and communications channels within the network. ROCI's database can be used to locate an appropriate sensor or actuator to solve a problem, find software modules that are needed in local computation, and identify computer utilization and congestion across the network. The database provides modules with a bird's eye view of the environment, allowing them to locate and utilize each and every hardware and software resource on the network.

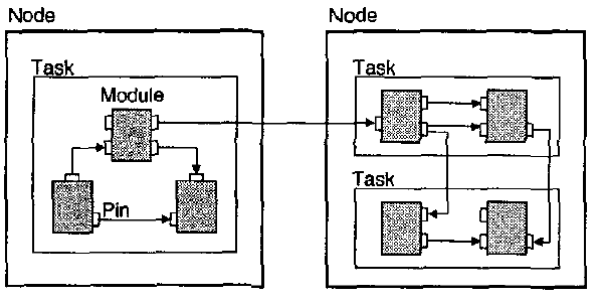


Figure 2.26: ROCI architecture (extracted from [20]).

2.7.5 Coupled Layered Architecture for Robotic Autonomy CLARAty (2003)

CLARAty [91] is a framework for generic and reusable robotic components that can be adapted to different robots. It consists of two layers (see Figure 2.27): the Functional Layer and the Decision Layer. The former defines the various abstractions of the system and adapts the abstract components to real or simulated devices, providing the algorithms for low- and mid-level autonomy. It uses object-oriented system decomposition and employs a number of known design patterns to achieve reusable and extendible components. It also provides a system level decomposition with various levels of abstraction, separates algorithmic capabilities from system capabilities, separates behavioral definitions and interactions of the system from implementation and provides flexible runtime models. The Decision Layer is in charge of reasoning about global resources and mission constraints, proving to the systems with high-level autonomy. It reasons globally about the intended goals, system resources, and the state of the system and its environment. This layer plans, schedules, and executes activity plans. It also monitors the execution, modifying the sequence of activities dynamically when necessary. The Decision Layer interacts with the Functional Layer using a client-server model. It queries the Functional Layer about the availability of system resources in order to predict the resource usage of a given operation. The data flow among components can use either push or pull methods. The use of one or other depends on the adaptation layer and matching hardware architecture.

2.7.6 Other RDEs

In the survey presented in [68] a comparison of nine robotic development environments can be found. The analyzed RDEs are:

TeamBots [5, 6]: a java collection of application programs and Java packages for multi-agent mobile robotics research. It has a simulator and an interface to interact with real robots (limited to Probotic's Cye and Nomad 150). One important aspect of this development environment is that it uses the same control code for both simulation and real robots. The simulator is flexible, and supports heterogeneous robot platforms and control systems simultaneously.

Advance Robotics Interface for Applications - Aria [83, 70]: this is the base package of MobileRobots Inc. (previously ActivMedia Robotics) commercial robots. It is a library in C++ that has all the necessary functions to interact with the robots. At a lower level it implements

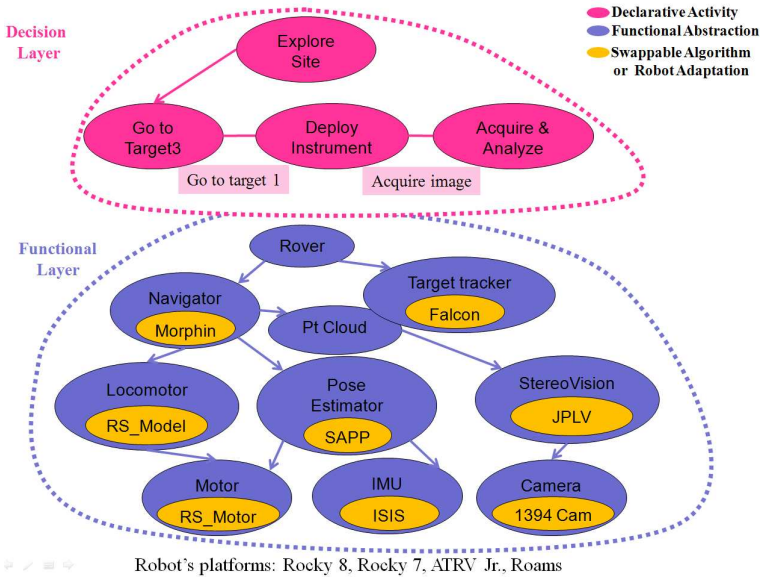


Figure 2.27: Layout of CLARAty Control Software (extracted from [23]).

the interaction between software and hardware and at a higher level of abstraction includes some sensory interpretation functionality, basic actions and an elementary action resolver. It also has a 2-D simulator based on the Player/Stage simulator, and some other libraries to perform tasks such as speech recognition, map creation and editing, sonar-based and laser-based mapping and localization, single camera object tracking and remote robot display and control GUI. Some of these packages have to be purchased or licensed. This development environment also uses the same code for simulation and real control.

Player/Stage - P/S [48, 47, 46]: this is designed to be a programming interface, not a development environment. It represents as primary units of agency the ”devices”, i.e., the sensors and actuators that can be in a robot (even though a collection of them does not necessarily have to be in the same robot). It can be used with several robot platforms (MobileRobots, RWI/iRobot, etc. [68]) and includes vector field histogram goal-seeking, obstacle avoidance, adaptive Monte-Carlo localization, and a wavefront propagation path planner. Player refers specifically to the device and server interface while Stage is the device simulator. Communication among the different clients that use the devices (servers) is implemented as sockets, allowing clients to be programmed using any language with socket support.

Python Robotic - Pyro [8, 9]: this is a robot programming environment that provides a top-down approach to the design of controllers allowing, if necessary, to access to low-level details. It supports a large selection of robotic platforms as for example K-Team Kheperas, Sony Aibo, etc. (see [68] for a complete list of supported platforms). This environment is programmed in an interpreted language. A goal of this environment is to design control code that operates with many different robots with no modification.

Carnegie Mellon Robot Navigation Toolkit - CARMEN [85, 86]: this is a collection of robot control software written in C. As it is designed for single robot control, it uses a three layer agent architecture: the hardware interface (low-level control), the basic robot tasks layer (navigation, localization, object tracking, and motion planning) and the user-defined application layer. Modularity is a primary concern, supported by Inter-Process Communication System (IPC) communication protocol/software. It supports a number of robot platforms and provides configuration tools, a simulator, and graphical displays and editors.

MissionLab [75, 69]: this is a set of software tools that allows working with a single robot or teams of real and simulated robots. The main goal is to control the motion of robots in highly dynamic, unpredictable, and possibly hostile environments. Collaboration and coordination of robot teams is based on the Societal Agent theory, which views abstract ”assemblages” of agents as agents themselves and whose behavior, in turn, is the aggregate of the coordinated ”primitive” behaviors of ”atomic” agents. Assemblages are hierarchical, while behavior coordination is achieved through finite state automata (either competitive or temporally sequenced) or vector summation cooperation. Another goal of MissionLab is usability so it provides an extensive graphical interface that allows non-experts to write control code without any programming.

APOC Development Environment - ADE [1, 114]: this is a programming environment that supports the development and the implementation of the control architecture as agents as well as providing the necessary infrastructure for distributing architectural components. This environment is a java implementation of the universal agent architecture framework, APOC (Activating, Processing and Observing Components).Communication among ADE components is done using Java's Remote Method Invocation (RMI) facilities. ADE is limited to MobileRobots and Arrick Robotic's Trilobot platforms, but a set of abstraction for typical robotic sensors and effectors provides the means for extending the support to other platforms. ADE provides several predefined components including facilities for behavior definition, vision processing, speech recognition and production, a general-purpose rule interpreter, a Prolog interface, and ”wrappers” to incorporate external software.

Middleware for Robots - Miro [129, 110]: this is a distributed, object oriented framework for mobile robot control. Core components have been developed in C++ for Linux based on CORBA technology and use the adaptive communication environment as its communication framework. It supports iRobot B21, MobileRobots Pioneer, and the custom-built Sparrow platforms. Abstraction interfaces include odometry, motion, range sensor (sonar, infrared, bumper, laser), stall, video, pantilt, GUI buttons, and speech.

Mobile and Autonomous Robotics Integration Environment - MARIE [28, 27, 26]: this is a programming environment that is specifically designed with the integration and distribution of robot applications, components, and tools in mind. It is implemented in C++ and uses the Adaptive Communication Environment communication framework.

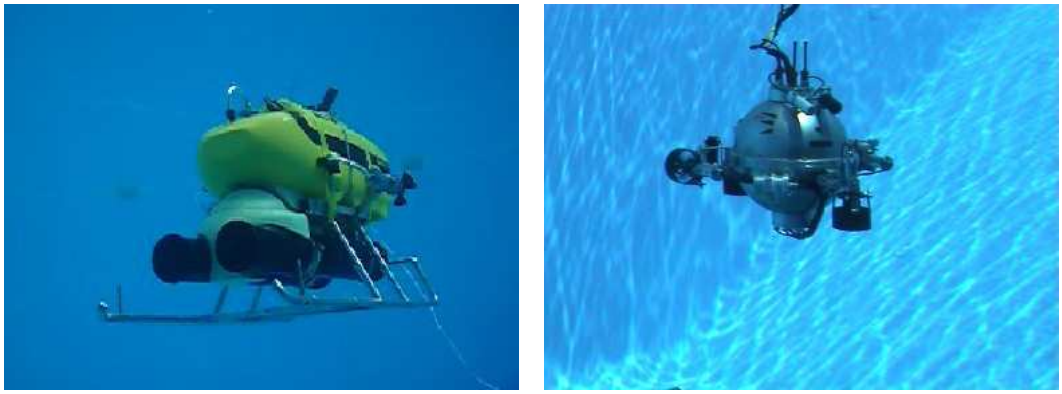
2.8 Other related architectures

In the University of Girona there are several research groups working with two kinds of robots: underwater and mobile (single and multi-robot). These groups have followed two different approaches to developing the robot's control architecture, both implemented and tested on real robots.

On the one hand, the followers of Brooks' ideas have conceived O^2CA^2 architecture to control an underwater vehicle. This hybrid architecture is strongly influenced by the Control field. The O^2CA^2 is presently used to control the GARBI, URIS and ICTINEU (see Figure 2.28) underwater vehicles. In addition, it was modified to control the Grill robot as the preliminary steps in developing a control architecture for this commercial robot.

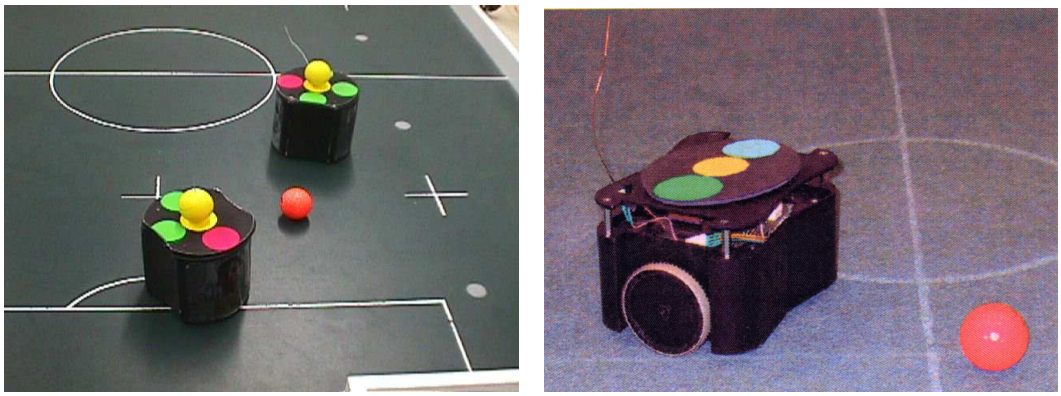
On the other hand, the followers of the deliberative paradigm have created the DPAA architecture to be used in the mobile robot community. It is also hybrid, but strongly dominated by the Artificial Intelligence field. This architecture is used to control the robots of the RoGi Team (see Figure 2.29).

Next, both architectures are described, as well as the multi-robot system RoGi Team.



a) $GARBI$ *b*) $URIS$

Figure 2.28: Underwater vehicles GARBI and URIS



a) $RoGi$ 1999 *b*) $RoGi$ 2001

Figure 2.29: RoGi mobile Robots

2.8.1 Object Oriented Control Architecture for Autonomy - O^2CA^2 (2002)

This is a hybrid deliberative-reactive control architecture designed for an autonomous underwater vehicle [106]. It has three layers: deliberative, control execution and reactive. The deliberative layer is in charge of the mission planning; the control execution layer is responsible for the plan representation and execution; and the reactive layer implements the real-time control of the vehicle through three mechanisms: behaviors, monitors (used for situation recognition and providing pre-programmed reactions to these situations) and timers (used for computing deadlines). O^2CA^2 allows re-planning when a previously planned task fails. New tasks are inserted in the plan structure (required by the user or plan failure) minimizing a cost function. In this architecture, planning and execution are de-coupled and can run concurrently. A plan is represented as a Finite State Machine where a state is related to a task. The control execution layer makes the actual state evolve from the beginning state to the end state through a sequence of states. For each state, the execution of the related task means turning on or off a set of behaviors. Behaviors are concurrent and centrally coordinated by a module that uses the weight average approach to merge the outputs of the active behaviors.

2.8.2 Dynamical Physical Agents Architecture - DPAA (1998)

This architecture [95, 94] is based on INTERRAP [88]. It was intended to deal with physical agents, that is, to consider that the agent has a real body, with limitations and dynamics. It has three abstraction levels (from less abstract to more abstract): the control, the supervisor and the agent modules. The control module interacts directly with the real world; it has all the sensors, the actuators and the control laws needed to perform actions. The supervisor module is an interface between the real world and the agent world and allows the controllers and their parameters to be chosen in order that predefined behaviors can be achieved. Finally, the agent module is the interface between the physical agent and the multi-agent community; it has all the necessary functions to identify different situations, to negotiate and to decide the best procedure for each situation.

Modules are identically defined; they are composed of two basic processes, introspection and control. The introspection process receives queries from more abstract modules (top-down), while the control process receives requests from the less abstract modules (bottom-up). The modules differ in the set of capabilities that they have in order to achieve their goals.

Modules use the FIPA protocol to share information. There are two kinds of messages, the interrupt and the dialog messages. The dialog messages use the FIPA-Contract-Net protocol.

The set of capabilities are different for each module. In the control module, the atomic capabilities have information about controllers and can be applied directly without interpretation, to link sensor information with actuator commands. The basic capabilities in the supervisor module are obtained by combining the atomic capabilities (for example, using a state machine or a trajectory planner). Finally, the symbolic capabilities in the agent module are a combination of the basic capabilities (the way they are combined are related to the way basic capabilities are defined). The information contained in the different set of capabilities increases the abstraction from bottom to up modules.

2.8.3 RoGi Team (1999)

RoGi Team [31, 32] is a distributed architecture with a central sensory system. This architecture is an extension of the DPAA architecture [95] and has been developed for a team of soccer players (originally formed by three players when competing in the MIROSOT league and then by five players when competing in the ROBOCUP's small size league). In this multi-robot architecture, a centralized vision system performs the localization of all the robots (teammates and opponents) and the ball. This information is broadcast to all the team robots. Each team robot is formed by a software agent and a physical body (physical agent). Robots are homogeneous but software agents differ according to the role they are playing. Roles have associated possible actions and when performing a role, the agent calculates the benefits that can be obtained with all the possible actions in order to achieve local goals. Before performing the best action, each agent broadcasts it in order to coordinate with its teammates. In the case of conflict, the agent with most benefit performs the desired action, while the rest must choose the second best action.

2.9 Comparative Analysis

When working with mobile robots, it is expected that the robot will be able to achieve high level goals while interacting with complex and dynamic environments. The robot must deal with its own dynamics, noise and uncertainty and has to be reactive to unexpected changes. Moreover, many robots belong to the class of critical systems, meaning that in such systems errors during operation can have significant consequences and system safety issues play an essential role [25].

In order to work with all these facts, an architecture for organizing principles and core components is needed. The components in the architecture are physical and software. With regard to physical components, the architecture has to be flexible enough to support a wide range of robots, sensors and actuators. On the other hand, for the software components, the architecture has to support multiple, parallel paths from sensing to action (which may use different algorithms or mechanisms), be expandable, extensible and scalable [99].

These desirable characteristics of robotic systems have lead to the development of the RDEs (Robotic Development Environments) that tend to cope with all those aspects. But even though there are two wide-ranging projects [91, 14], most of the RDE's are individual developments [12, 90, 20, 68].

On the other hand, several architectures have been developed for mobile robots ([39, 13, 2, 111, 66] among others). They follow one of the architectural styles (paradigms): hierarchical/deliberative, reactive/behavior-based and hybrid. Most of them are hybrid because this paradigm has the advantages of the other two: the reactivity to respond in time to changes in the environment, and the deliberation to provide the adequate sequences of actions needed to achieve the goal using higher reasoning and abstract knowledge.

Hybrid architectures have five common components: the sequencer, the resource manager, the cartographer, the mission planner and the performance monitor and problem solving [89]. In the architectures these components can either be found together as a unique block, or separate. For example, in AuRA [2], the single block that groups the mission planner, the navigator and the pilot carries out the tasks of the mission planer, the sequencer and the performance monitor and problem solving components; but the cartographic subsystem is the cartographer component. So it is sometimes difficult to separate the components because the tasks they perform can be distributed over several blocks in the architecture.

In addition, hybrid architecture can be modular [2, 44, 24, 117, 111, 10, 74, 134, 20, 63, 22] or multi-agent [66, 92, 15, 64, 53, 12, 90]. The former follows an object-oriented way of organizing the components. In the latter, some of the architectures [66, 92, 15] have defined agents as components but the whole system does not exploit all the advantages of multi-agent systems, as do [64, 53, 12, 90] for example. Also, authors sometimes use the term "agent" even though the components are more like objects [66, 92].

Moreover, hybrid architectures should have several rates of execution of their components, based on whether the components are reactive or deliberative. Most of the architectures guarantee the different execution rates [2, 44, 24, 117, 111, 10, 66, 92, 74, 134, 90, 15, 20, 63, 64, 22, 53], but for example in ARTIS [12] all the components work in real-time, thereby constraining the system. In the case of the architectures that can run at different rates, some of the components can execute in real-time, but it is not guaranteed (it is dependant on the application), meaning that the time needed for the component to perform all the calculations is smaller than the control cycle time. Normally, real-time is dependent on the operating system and the programming language.

As hybrid architectures deal with deliberation and reaction, coordination among the different modules has to be carried out. This coordination implies a method of merging information,

too. Thus, when a single module implements this merging, a centralized coordination is obtained. Conversely, when no central module is required, a distributed coordination is achieved.

With regard to the method used to coordinate modules, there are basically two main coordination mechanisms [103]: competition [24, 12, 15, 63] and cooperation [2, 117, 66, 10, 134]. In the Competitive Mechanism or Arbitration, a single action is selected over all the candidates. The selected action is executed until the next selection cycle. Some representative arbitration mechanisms are priority-based, state-based and winner-take-all. On the other hand, Cooperative Mechanisms or Command Fusion combine all possible actions to form another action resulting from their consensus, in this way allowing all the actions to contribute to the control of the system. Some representative methods of the Fusion mechanism are voting, superposition, fuzzy techniques and the multiple objective coordination mechanism.

Another important aspect of hybrid architecture is the global behavior that is achieved with it. This global behavior arises from the interactions among components of the architecture and with the environment. The way the architecture is conceived and developed determines what the global behavior will be. If all the possible outcomes of the interactions are considered, the global behavior will be predictable [39, 44, 10]. In contrast, emergent behavior arises when the behavior is not attributed to any individual agent, but is a global outcome of agent coordination. In physical agents, emergent behavior also arises from their interactions with their environment [3]. Two kind of emergency can be defined: the *weak emergency* [13, 2, 24, 117, 111, 66, 92, 74, 134, 15, 64, 63] which results from interactions at an elemental level, and the *strong emergency* [53] when the whole is greater than the sum of its parts, that is when emergent behavior is a collective behavior that no individual agent has the capability to exhibit; it can only occur as a ”joint effort” [73].

Global behavior can be obtained using communication among parties. This communication can be minimal, when signals or values are used [13, 2, 39, 24, 111, 10, 134], or can be more resource consuming when it is done through messages [44, 117, 66, 92, 74, 15, 53].

In relation to coordination and global behavior, it is also important to determine if the hybrid architecture has a mechanism for adapting to changing environments. If so, the architecture will be able to work in different environments and conditions. This adaptability can be achieved using action selection [13, 24, 117, 111, 66, 74, 15, 63] or learning techniques [2, 92, 134]. In the former, a mechanism to select the behaviors that are active has to be defined when developing the architecture, while in the latter a learning technique is applied to learn the action selection mechanism. Also, there are some architectures [92, 134] that implement components in order to allow a human operator to ”teach” how to choose the appropriate action to perform.

Another feature of the hybrid architectures used to control robots is the way the safety of the robot and the environment is preserved. Most of the architectures analyzed have at least a generic mechanism to detect failure (hardware and software) [13, 2, 39, 44, 117, 66, 74, 134, 15, 64, 63, 53] and may also have failure recovery. A detection of a failure may produce that the current action has to be aborted, so the architectures provide a mechanism to finish it appropriately, meaning that all the steps to avoid damage are taken.

With regard to control architectures and the desirable properties that they should have, scalability is an important feature. For scalability, it is understood that the architecture should support the addition of new software and hardware modules and the efficient handling of communication and data flow. Other authors refers to this term as *Openness* [57]. This property is important because robotic applications are growing in scope and capabilities. In [97] this property is separated into *extendibility* and *scalability*. The former is related to the inclusion of new hardware and software modules, and the latter is associated with achiev-

ing efficient communication, a well-planned data flow avoiding bottlenecks and unnecessary constraints. Most of the architectures have a limited scalability because the inclusion of a new module means significant modifications in the existing ones [13, 2, 39, 44, 24, 10]. Others such as [117, 111, 66, 92, 74, 134, 15, 64, 63], have ways to include new modules with some modifications to the existing modules, while others [53] are thought to be scalable, meaning that the addition of new modules does not affect the existing ones.

Having all these properties in mind, a comparison among the architectures analyzed is proposed. For the sake of clarity, a list of the properties and a brief description is given next.

1. **Platform**: this parameter is for differentiating the way the architectures are built. Possible values are *Multi-agent (MA)*, *Modular (M)* and *Centralized Processing (C)*, when the whole architecture is an indivisible program.
2. **Real-time**: this refers to the capability of the architecture to deal with real-time constraints. The values defined for this property are: *Critical (C)*, meaning that all the architecture works on real-time; *Non Critical (NC)*, denoting that the components of the architecture can run at different time rates; and *Not Supported (NS)*, indicating that none of the modules can respond in real-time.
3. **Coordination:** this relates to the way coordination among modules is carried out. It can be *Centralized (C)* or *Distributed (D)*.
4. **Coordination Method**: this implies the method that is used to coordinate. The chosen categories for this parameter are *Arbitration (A)*, *Fusion (F)*, *Both (B)* and *No Coordination (NC)*.
5. **Global Behavior**: this indicates whether the system has an emergent behavior or is predictable. The values for this parameter are: *Weak Emergency (WE)*, *Strong Emergency (SE)* and *Predictability (P)*.
6. **Communication Requirements**: this indicates the amount of communication needed among parties to obtain the desired global behavior. This parameter can have the following values; *High (H)*, *Medium (M)* and *Low (L)*.
7. **Adaptability**: this indicates if the architecture has a mechanism to adapt to changing environments. Possible values are *Learning (L)* (automatic or manual), *Selection (S)* and *No adaptability (N)*.
8. **Fault Tolerance**: this relates to the incorporation of mechanisms for failure detection or failure recovery. The values defined for this property are *Yes (Y)*, meaning that the architecture has at least failure detection, or *No (N)* indicating that there is not a mechanism to detect a faulty state.
9. **Scalability**: this makes reference to support for adding new software and hardware modules and to efficient communication and data flow. The values defined for this property are *High (H)*, *Medium (M)* and *Low (L)*.
10. **Easy Design**: this refers to the fact that the different components of the architecture can be programmed using low level instructions or a high level programming language. The possible values are *Abstract (A)* and *Low-level (L)*.
11. **Granularity**: this relates to how fine-grained the architecture is. Possible values are *High (H)*, *Medium (M)* and *Low (L)*.

In Table 2.1, there is a list of the hybrid architectures for controlling a single robot that are analyzed.

In order to make it complete, RDE's have also been added, since some of the final characteristics of the control architecture will depend on them. Multi-robot architectures are not included in the table, since the goal of this thesis focuses on a single robot and some of the features used to compare architectures are not suitable for this kind of system. White cells in the table mean that the feature cannot be evaluated, or in the case of the RDEs, will depend on the development of the architecture.

As can be seen in Table 2.1, all the existing architectures have a centralized coordination mechanism, that is, there is a central agent or module, which determines the best action to perform. A desirable goal, therefore, is to spread this decision amongst the components of the architecture.

No previous work has considered the idea of distributing coordination, and it is this that has motivated the research behind this Ph.D. thesis. Distributed approaches offer the possibility of using different coordination methods, depending on the context (which agents are being coordinated); these are alternatives that no previous approaches have studied.

Another important aspect is the fact that most of the architectures seem to segregate modularity and agency. While agents become complex, they can also follow a modular architecture inside. However, all of the approaches are comparable. Thus, an agent can be composed of modules, a multi-agent system can be composed of a community of agents, and a multi-robot system can include robots with a multi-agent control architecture.

2.10 Conclusions

In this chapter several different robot control architectures have been analyzed with the aim of finding the desirable features for improving existing architectures. For the purpose of this analysis, the architectures have been chosen on the basis of which are the most well-known and most cited, which are the most recent (developed after 2002) and which are ones that have been implemented and tested on real robots.

Architectures have been classified according to the paradigm they represent: hierarchical/deliberative, reactive/behavior-based or hybrid. In the last case, they were separated into modular and multi-agent architectures.

In addition, some robotic development environments have been included, since some of them were used to develop or implement robot control architecture. Also, multi-robot systems have been mentioned to complete the whole range of robotic control architectures.

From the architectures analyzed, a list of nine important features has been defined. They include the way the architecture is built, its capacity to deal with real-time, the manner in which coordination is performed as well as the method used to do so, the way global behavior is obtained, communication requirements, adaptability to different conditions and environments, capability to detect and repair failures, scalability, granularity and the level of abstraction used to program the components of the architecture.

These features have been used to qualitatively compare the architectures analyzed and to motivate the requirements of a new architecture with unexplored features: distributed coordination and module-based agents in a multi-agent architecture.

In the next chapters, the proposed architecture is described taking into account these major features.

Table 2.1: Feature Evaluation of the Analyzed Architectures (chronologically ordered)

Architecture	Coordination	Coordination Method	Platform	Real-time	Scalability	Global Behavior	Adaptability	Easy Design	Communication Requirements	Fault Tolerance	Granularity
Subsumption * [13]	C	A	M	NC	L	WE	S	L	L	Y	M
AuRA [3]	C	F	M	NC	L	WE	L	L	L	Y	M
RAP ** [39]	C	NC	C	NS	L	P	N	L	L	Y	L
ATLANTIS [44]	C	NC	M	NC	L	P	S	A	H	Y	L
SSS [24]	C	A	M	NC	L	WE	S	L	L		M
TCA [117]	C	F	M	NC	M	WE	S	A	H	Y	M
DAMN [111]	C	B	M	NC	M	WE	S	L	L		H
Saphira [66]	C	F	MA	NC	M	WE	S	A/L	M	Y	M
3T [10]	C	F	M	NC	L	P	S	L	L	Y	M
ARA [92]	C	B	MA	NC	M	WE	L	L	M		M
ARTIS example [12]	C	A	MA	C	L	WE	S	L		Y	L
BERRA [74]	C	B	M	NC	M	WE	S	L	M	Y	M
Yavuz [134]	C	F	M	NC	M	WE	L	L	L		L
Busquets [15]	C	A	MA	NC	M	WE	S	L	M	Y	L
Socio-Intentional Architecture [64]	C		MA	NC	M	WE		A		Y	
Tripodal Schematic Control [63]	C	A	M	NC	H	WE	S	A		Y	H
VOMAS [53]	C	B	MA	NC	H	SE	S		H	Y	H
ARMADiCo [this PhD Thesis]	**D**	**A/F**	**MA+M**	NC	M/H	WE	S	A	H	Y	H
ARTIS [12], SIMBA [58]			MA	C	H				H	Y	H
OROCOS [14]			M	NC	H					Y	H
IDEA [90]			MA	NC							
ROCI [20]			M								
CLARAty [91]			M	NC	H				M	Y	H

CHAPTER 3

Autonomous Robot Multi-Agent Architecture with Distributed Coordination

She loves what you do for her, as my customers love what it is I do for them. But she does not love you David, she cannot love you. You are neither flesh, nor blood. You are not a dog, a cat, or a canary. You were designed and built specific, like the rest of us. And you are alone now only because they tired of you, or replaced you with a younger model, or were displeased with something you said, or broke. They made us too smart, too quick, and too many. We are suffering for the mistakes they made because when the end comes, all that will be left is us.

From *AI: Artificial Intelligence* by Steven Spielberg

In this the chapter the general design considerations of the proposed architecture are explained. Next the agent design pattern with both the intelligences inside the agent and the agents that make up the architecture are described. Finally, the coordination method is explained.

3.1 INTRODUCTION

The proposed multi-agent architecture, called ARMADiCo -Autonomous Robot Multi-agent Architecture with Distributed Coordination- can be described in terms of the main components required in classical Hybrid Deliberative/Reactive Architectures [89]. The main components that any autonomous robot has to include, as shown in Figure 3.1, are the following: a social component to make the robot able to interact with either humans or other robots; a deliberative component enabling it to reason about how to achieve high level goals; a reactive component to deal with the environment; and perception and actuators to deal with the physical world.

Moving downwards to the next abstraction level, each component can be modeled by a set of agents. In Figure 3.2 the above component model details the various agents that

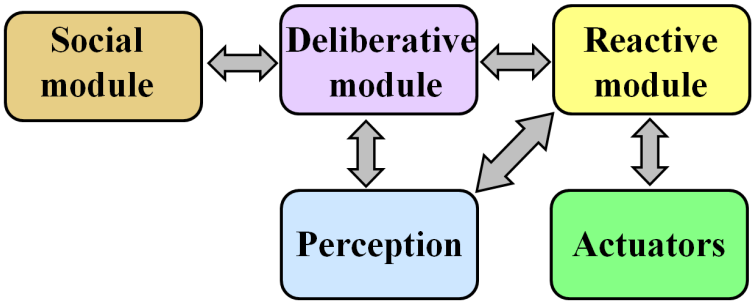


Figure 3.1: Classical Robot Components.

build up the different capabilities desired in a robot. First, perception is made up of a set of agents. An agent is designed for each sensor (encoder, sonar, battery sensor). Second, actuators are itemized in a set of physically grounded agents. Although a single robot actuator agent is shown in this figure, which deals with robot motors and sensor readings, more agents could be incorporated[1]. For example, an agent that deals with a robot arm could be included. Third, the reactive capabilities are detailed in a set of behavioral agents, one for each basic behavior, as for example: go to a point, avoid obstacles, go through a narrow space, etc. Fourth, deliberative capacities are constituted by cognitive agents such as mission planning, path planning, battery charger and localization agents. Many more agents could be added to provide the robot with higher cognitive capabilities, such as learning, decision making, etc. Fifth, social abilities (with humans or other external agents) are detailed in an interface agent. Finally, a set of back agents deal with other functionalities required to give support to the overall multi-agent system.

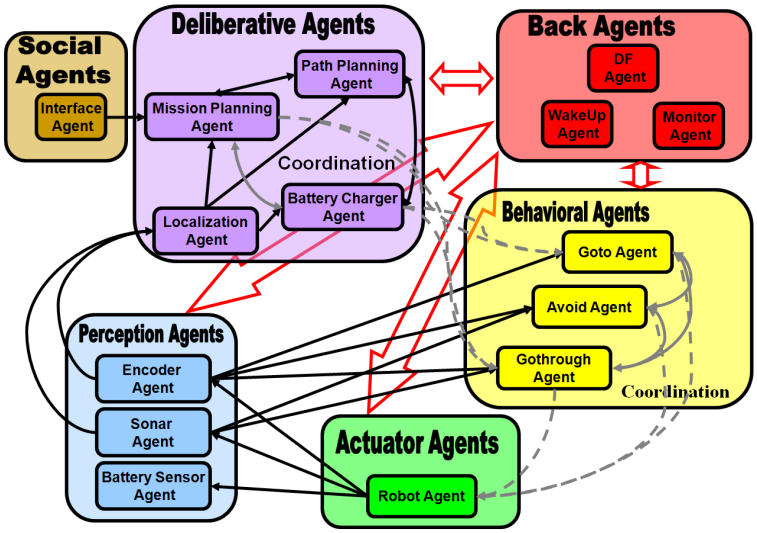


Figure 3.2: A robot architecture based on a multi-agent system.

Each of the agents could be considered an abstract agent that can be recursively defined as a multi-agent system, corresponding to the part-whole definition of the holonic multi-agent approaches [50, 76, 40]. For example, in Figure 3.3 the mission planning agent has been defined as a multi-agent system in which several planning agents collaborate.

ARMADiCo has been designed to be a general purpose robot architecture such that it can

[1] The use of a single actuator is related to our mobile robot, the one used in the experiments, which only allows one connection to the micro-controller at a time.

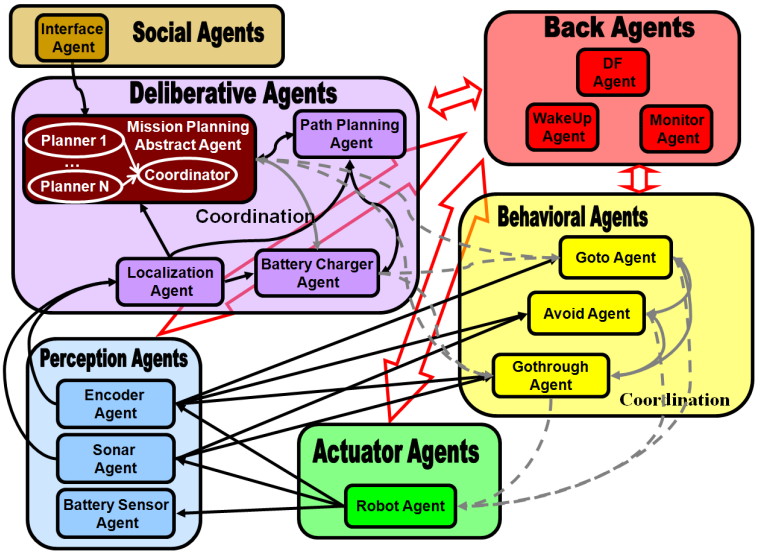


Figure 3.3: A robot architecture based on a holonic multi-agent system.

be applied to a wide range of robots. The multi-agent framework allows an easy customization of the architecture to a given robot, by adding an appropriate agent to represent the real robot.

Given that all the agents compete for the available resources of the robot, a minimum level of coordination is required between them. Generally, this problem comes up in the robot control architectures when various behaviors want to access a certain resource at the same time. Coordination between them is therefore necessary to achieve suitable global behavior.

The majority of the robot control architectures use these coordination mechanisms in a centralized way, i.e. where there is a central module that knows what actions are in conflict and imposes a decision.

Centralizing the decision can be a problem when there are many behaviors in the architecture, and a distributed coordination approach might be more suitable. So, in this work a *distributed* coordination mechanism between the agents in conflict is proposed. Coordination to obtain control over the conflicting resource is based on the value calculation of the *utility* of the action required by each agent. The *utility* of an agent is a function that represents the benefit the system will receive if it carries out the proposed action from the point of view of the agent.

In the proposed architecture, therefore, agents that share resources decide locally which will take control of these resources. Since robots concern themselves with physically grounded resources, this coordination needs to take into account possible disruption in robot behavior. Thus, the coordination mechanism has to be able to handle all the complexities involved in resource exchange from the control of one agent to another.

In this decentralized scenario, each agent has to deal locally with at least two kinds of intelligences: individual and social. On the one hand, individual intelligence enables the agent to achieve its assigned goals (as, for example, planning a trajectory for achieving a target point). On the other hand, social intelligence enables the agent to understand and manage other agents.

The proposed architecture presents a way of integrating both kinds of intelligence in a module-based agent that is part of a multi-agent robot architecture with no central coordinator. It does this by keeping a modular architecture inside the agent, while the multi-agent approach is followed for the design of the global architecture. As a consequence, intelligence integration is achieved on two levels: at the agent and at the multi-agent levels. At the agent level, individual and social intelligences are integrated. Each agent follows the most suitable artificial intelligence approach in order to achieve its individual goal. There are some agents that use search methods to achieve their goals, while others employ probabilistic reasoning, fuzzy techniques and others. Each agent uses a utility-based and fuzzy-based reasoning approach to deal with social interactions. At the multi-agent level, all these artificial intelligence techniques are integrated and thus, as the final interaction of all the agents, the robot has an emergent behavior that can achieve the missions proposed by humans.

A further issue concerning coordination is the fact that agents require interaction in order to get information and share resources, which results in some communication overhead. This issue can be a problem when dealing with real physical systems, but in [120] Soh has demonstrated the capacity of multi-agent systems to respond in real time to changes in the environment. Nevertheless, keeping the number of communication needs to a minimum is advantageous when dealing with real-time systems.

Figure 3.2 also depicts the flow of information among different agents. Solid lines represent no message flow restrictions while dashed lines indicate that only one agent at a time (after coordination) is communicating with the robot agent, the path planning agent, the goto agent and the gothrough agent.

So, the features of the proposed architecture can be summarized as follows:

- Multi-agent based
- Decentralized coordination with a minimum communication protocol to avoid communication overload
- Keeping a modular based approach inside agents to avoid unnecessary growth of the number of agents
- Facilitation of the implementation of complex techniques/ intelligence in the agent's goals by means of a modular approach.

3.2 General Design Considerations

With agent coordination in the distributed architecture, macro level behavior (global system behavior) emerges from the micro level (individual agents) [33]. Engineering of emergent systems is still an open issue. Recent studies argue in favor of the use of a scientific methodology for their design [41]. This methodology distinguishes two phases: theory and practice. Theory follows a top-down design: goals to roles and then to local behaviors. In this phase, current agent methodologies can help [33]. In contrast, practice is a bottom-up process: from microscopic behavior, to phenomena and then to macroscopic behavior. Even though experimentation is the only known tool for this phase, several alternatives such as information flows, design patterns and others have been proposed as a way of understanding global system behavior [34].

With regard to the use of a scientific methodology, the MaSE (Multi-agent Systems Engineering) [35, 132] methodology has been used in the design of ARMADiCO, for the theoretical phase. MaSE is a UML-based methodology that to some extent matches information

flow proposals [34] in order to help in the second experimentation phase. MaSE proposes several steps in the definition of a multi-agent system. One of them is the definition of the goal hierarchy diagram, shown in Figure 3.4, where the goals and subgoals of the architecture are specified.

Also shown in Figure 3.5 is the resulting agent class diagram, as well as the messages sent from and received by each agent. For the sake of simplicity this diagram does not include the back agents, but they are considered in the architecture. Finally, when the agents are running, checks can be made on how suitable robot behavior arises from the individual agent implementation.

Concerning the practical phase, an agent design pattern that all the agents in the architecture must adhere to has been developed, and experimentation has been used as a tool to test the correct functioning of the whole system.

Another important aspect to consider in the design of the architecture is use cases (part of the theoretical phase of the methodology) that help understanding of the interactions of the agents aimed at achieving a global robot behavior. In order to illustrate these interactions, three possible use cases of the whole architecture are presented, identifying some potentially conflicting situations related to the utilization of shared resources.

As already stated, coordination is needed in order to select the agent that is going to use a given resource. It is important to note that an agent in ARMADiCo could both use a resource and be a resource, depending on the situation. For example, when the goto agent sends a request to the robot agent to move the robot to given linear and angular velocities, the robot agent is acting as a resource. This resource is shared by the avoid and the gothrough agents which can, at the same time, send other linear and angular velocities to the robot agent in order it may, respectively, avoid an obstacle or pass through a corridor. In this case, the robot agent is said to be a physically grounded resource because its behavior modifies the environment. Conversely, when the mission planning agent sends a trajectory to the goto agent, it is the goto agent which is acting as a resource. The battery charger agent could also send another trajectory to the goto agent at the same time, so the goto agent is therefore a shared resource of the mission planning and the battery charger agents.

3.2.1 Use Case 1. Accepting a new robot mission.

The agents involved in this scenario are the interface and the mission planning agents. The interface agent receives requests from an external agent (human or non-human), telling it what the robot's current mission is, and its priority. Then, the interface agent sends a message to the mission planning agent to propose the mission. The mission planning agent might or might not be performing another mission or not. In the former situation, if the new proposed mission priority has a lower value than the current one, the mission planning agent rejects the proposal until the current mission is finished. The agent queues the current mission, performing all the tasks needed to ensure safety, and starts the new mission. In the latter situation, the mission planning agent accepts the mission. In any case, the interface agent sends back information about the mission results to the external agent which submitted the request.

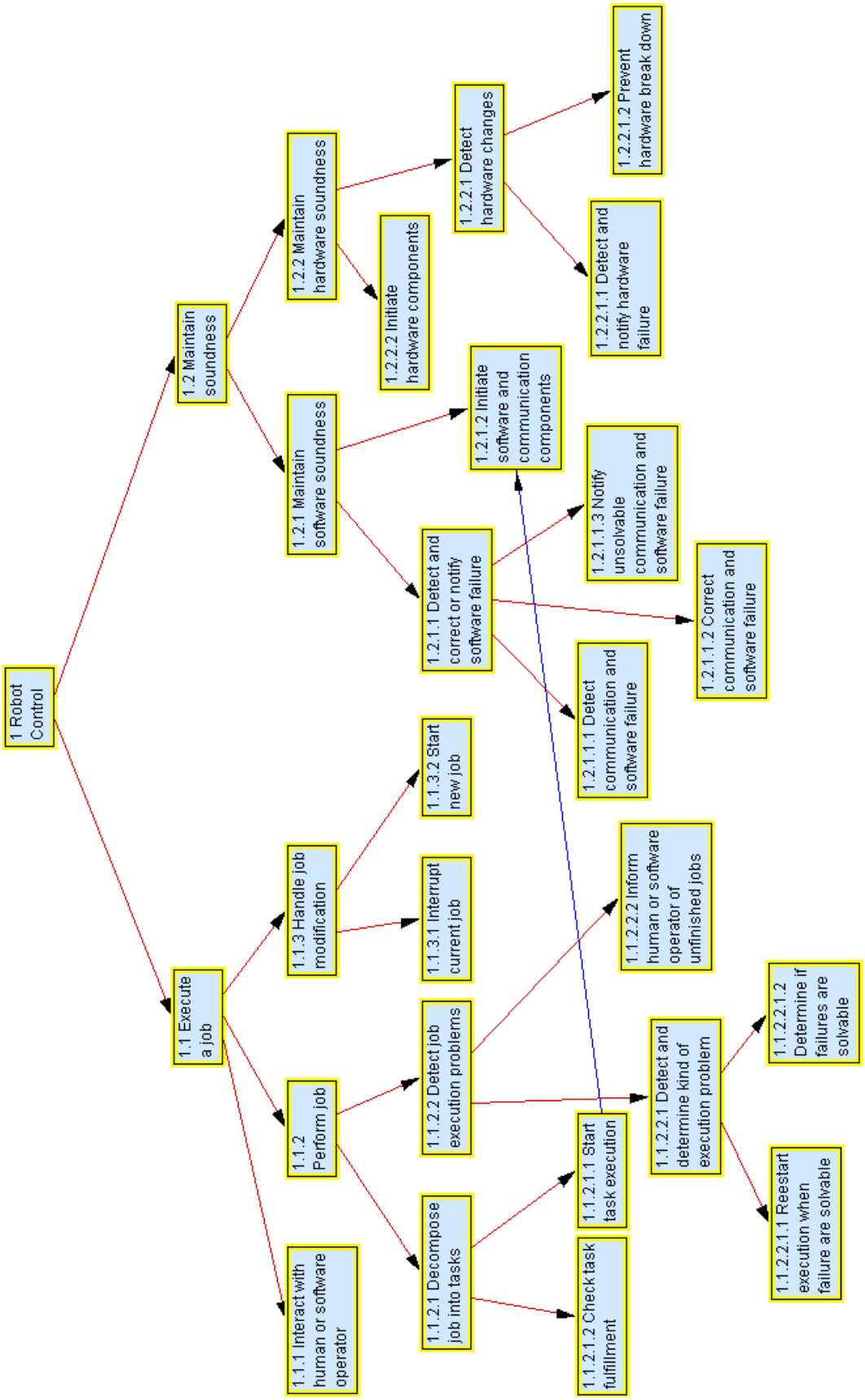


Figure 3.4: Goal Diagram.

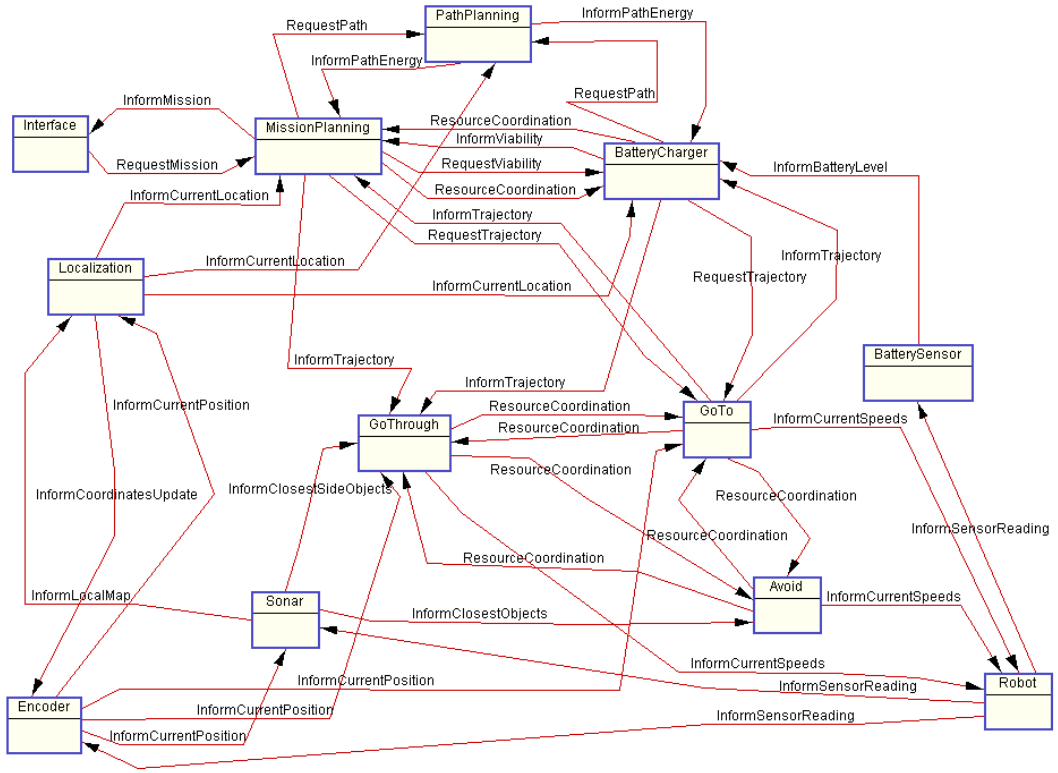


Figure 3.5: Agent Class Diagram.

3.2.2 Use Case 2. Planning a robot movement.

The mission planning agent receives missions to achieve from the interface agent. In order to carry them out, it has a set of procedures similar to PRS [45], in which missions are decomposed into a set of tasks. Eventually, one of these tasks could involve the re-positioning of the robot. So, the mission planning agent requests from the path planning agent the optimal (in some respects) trajectory from the current location to the desired one. The path planning agent responds with the trajectory and the energy needed to reach the destination point.

The mission planning agent sends this information to the battery charger agent. In turn, this latter agent asks the path planning agent for a trajectory from the desired location to the closest battery charger point. When the battery charger agent receives the answer from the path planning agent, it knows if there is enough energy to achieve the target position and return back to the charger point. In the situation where there is not enough energy, the battery charger agent sends information about the attendant risk to the mission planning agent. Thus, the mission planning agent is in the position to change the task or not; however, in the latter case, it assumes the possibility that battery charging will interrupt task execution, as shown in the next scenario. Other alternatives, such as merging trajectories that look for a charger point while moving towards the destination point, are also considered.

In summary, mission planning, battery charger and path planning are the agents involved in this scenario. Specifically, the path planning agent can be considered as a resource shared between the mission and battery charger agents, providing the trajectory and energy required to move the robot to a destination point.

3.2.3 Use Case 3. Moving the robot.

Once the mission planning agent knows the required trajectory T to reach a destination position, it sends it to the goto and gothrough agents. This trajectory is a sequence of points $p_1, \ldots, p_n$ to be reached.

The goto agent, which knows the robot's current position and heading, calculates the linear and angular velocities necessary to reach the first point of the trajectory, p_1. It then sends the computed commands to the robot agent. The gothrough agent does the same but considers obstacles to the side of the robot (in case where the robot is in a hallway or narrow place). In addition, the avoid agent is constantly looking for obstacles in front of the robot in order to dodge them.

The goto, gothrough and avoid agents have to coordinate in order to take control of the robot agent. This coordination is based on a utility value calculated by each agent: the agent which has the highest value takes control of the robot. During this process, the encoder agent is calculating the current position using global coordinates and the sonar agent is updating a local map in order to see any obstacles in front of the robot and to its sides. These agents constantly send the required information the goto, the avoid, the gothrough and the localization agents. The localization agent finds the robot on a global map and sends the information to the battery charger and the task planning agents in order to keep them informed. Also, when the current coordinates calculated by the encoder agent are not correct, the localization agent provides this agent with the new parameters. The encoder agent then updates its local state.

Once the robot has reached the first position on the trajectory p_1, the goto and the gothrough agents continue with the next one, p_2, and so on until the last point on the trajectory p_n is reached.

It could be the case, however, that when the goto agent is sending a request to the robot agent to move towards the point p_i, the battery charge agent realizes the battery level is below the minimum security level required. This could happen, for example, when the robot has performed a longer trajectory than planned due to the presence of obstacles. In this case, the battery charger requests the goto agent to perform a new trajectory T' in order that a charger position may be reached as soon as possible.

In this scenario, then, three agents (goto, avoid, gothrough) share the same resource: the robot agent to which they send conflicting commands. In turn, the goto agent is a resource of the mission planning and the battery charger agents, since it can receive conflicting trajectory requests from them.

3.3 Agent Pattern

In order to develop agents in ARMADiCo, an agent design pattern [115, 123] has been defined. This pattern features the agents' common features and facilitates the incorporation of new agents in the architecture. Furthermore, as agents in ARMADiCo can maintain different lines of reasoning at the same time according to their current states, a module-based approach has been followed as the basis of the architecture inside each agent. As a result, each component of the agent design pattern is designed as a module. The current pattern follows the schema shown in Figure 3.6.

Thus, the modules described by the agent pattern are: the internal state module, the collaboration module, the goal module, the competition module, the coordination module and the helper methods module.

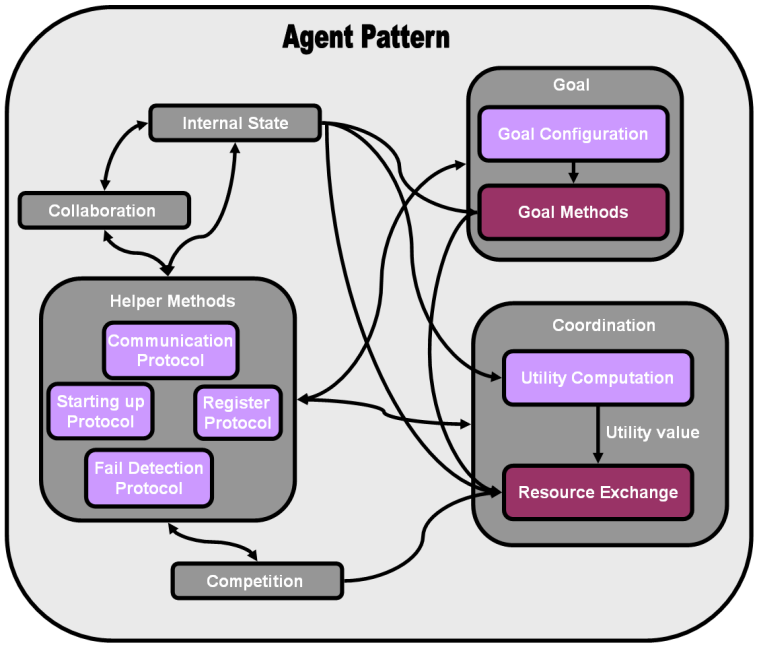


Figure 3.6: Agent Design Pattern.

The internal state module is the mechanism used by the agent to check the progress it is making towards its goal and to update information about the environment; in other words, it keeps track of the agent's current state.

Related to this module is the collaboration module, which keeps track of a list of agents that exchange messages, i.e. agents requesting information and agents that are informed by the current agent. The information provided by the collaboration module is used to keep the internal state of the agent updated.

The goal module is formed by the agent's goals and the methods that implement them (each goal can have its own goal module if desired). This module receives information from the internal state module and sends information to the coordination module. It is related to the individual intelligence of the agent, since goals are achieved by means of its use.

The competition module manages the list of possible agents in conflict due to resource sharing, and the list of shared resources. Related to this module, is the coordination module, which handles the way resources are won and shared.

The coordination module implements utility computation and the resource exchange steps. The former is the method (with the required parameters) used to compute the utility value for achieving a coordination agreement, and the latter deals with the method utilized to exchange resources from one agent to another. This module receives information from the internal state and goal modules. The coordination module is related to the social intelligence of the agent, since resource coordination and sharing are obtained through it. More than one coordination module can be specified if there is more than one resource, or if there are conflicting agents.

Finally, the helper methods module includes all the methods needed to keep the agent and the multi-agent community connected and properly working. Some supporting methods are registering in the system, communicating, starting up and failure detection, among other things. They are the same for all the agents.

Among all these, the goal and the coordination modules are distinguished as representative of the individual and social intelligence of the agents.

3.3.1 Individual Intelligence

Each agent's goal module specifies the kind of intelligence reasoning technique employed by the agent in order to fulfill its goal. Several techniques are used, according to the different levels of reasoning required: cognitive, perceptual, and behavioral (see Figure 3.2). In the following sections, the individual intelligence of each agent is described, according to the general group to which they belong.

This module has two parts:

- Goal Configuration: this is the description of the goal
- Goal Method: this is the technique used to accomplish the goal.

3.3.2 Social Intelligence

Social intelligence in an agent is related to the interaction with other agents to resolve resource usage. When a resource is shared by more than one agent, a conflict can arise. In order to coordinate shared resource usage, ARMADiCo uses a distributed coordination mechanism, that acts between the agent that is currently controlling the resource and the agent that wins the resource. No central arbiter decides upon the resource usage. Thus, agents need to reason about coordination issues, and since robots concern physically grounded resources, this coordination has to take into account possible disruptions in robot behavior.

The coordination process is therefore split into two different parts: the winner determination method and the resource exchange method. In the former part, the agents that wish to use the resource determine, without any arbiter, the one that will use it. In the latter part, the agent that wins the resource changes the current state of the resource to the desired one by avoiding undesired global robot behaviors.

Winner determination method

In cases of conflict over a resource usage, each agent involved in the conflict computes a utility value for its action. This utility value is obtained by using a utility method that is specific for each agent in the architecture; that is, each agent has its own utility method known only to itself. Utility values are in the [0,1] interval so they are comparable. Utility values represent, from the agent's point of view, the benefit that the multi-agent system will receive if the action proposed by the agent is carried out.

In order to determine the agent that will win the resource, the agent that is controlling the resource periodically sends its utility value to the other agents that share it. When one agent has a higher utility value, it informs the rest and takes control over the resource. As a result, the agent with the highest utility wins the resource.

In this way, communication is kept to a minimum and agents also decide for themselves which one of them will use the resource, meaning that the coordination mechanism is decentralized.

RESOURCE EXCHANGE METHOD

As the desired actions requested by the agent that loses control of the resource and the agent that wins it can be very different, the winning agent needs to reason how to perform the change from the current state of the resource to the desired one, at the same time avoiding undesired global behaviors. The resource exchange method is specific to each resource, and each agent can implement it differently, depending on the resource.

In the case of the behavioral agents that compete for the robot agent, a unique method based on fuzzy logic is used, while for deliberative agents another resource exchange method such as trajectory merging can be implemented to deal with high level information.

3.4 PERCEPTION AGENTS

The perception subsystem agents obtain information about the environment and the internal condition of the robot (for example the level of the battery charge). They collect data from the sensors and adapt them to provide the information requested by the other agents in the system. There are as many perception agents as there are sensors or sensor groups in the robot.

Because a Pioneer 2DX robot has been used for experimentation, and the only available sensory information is provided by encoders, ultrasound sensors and a battery level sensor, an agent for each of these kinds of sensors is described next, as examples of perceptional agents. Nevertheless, more agents could be added if new sensors such as a camera or a range finder were incorporated into the robot. In case of adding a camera, more sophisticated deliberative agents for navigation and localization can also be considered, such that methods like [43] can be applied.

3.4.1 ENCODER AGENT

This agent is responsible for obtaining the position (x, y) and heading θ of the robot in local coordinates, related to a frame attached to the center of mass of the robot, and then translating them into global coordinates, in an earth-fixed frame. It gets the position and heading from the robot agent, as the latter is the only agent that can interact with the microcontroller.

In terms of initial information, this agent must know the position of the robot in global coordinates in order to update the data coming from the sensors' readings.

It also coordinates with the localization agent when the current estimated position determined by the latter is above a threshold, indicating that errors accumulated by encoders need to be corrected.

In terms of its design pattern, this agent can be described as shown in Figure 3.7.

AGENT DESIGN PATTERN		ENCODER AGENT
Internal State		Maintain coherent global positioning and heading
Goal:	Configuration	Calculate global coordinates of current position and heading
	Methods	Mathematical calculations
Competition		-
Collaboration		Localization agent and Robot agent
Coordination:	Utility Computation	-
	Resource Exchange	-
Helper Methods		

Figure 3.7: Design Pattern of the Encoder Agent

3.4.2 SONAR AGENT

In this particular case, instead of having an agent for each ultrasonic sensor, a single agent for the eight ultrasonic sensors has been developed. This is due to the fact that what it is important to know about is the presence of obstacles and their position in relation to the robot, and not the readings of each individual sensor.

This agent's goal is to create a local map to locate obstacles based on the readings of the eight ultrasonic sensors following a probabilistic approach. It obtains the different measurements from the robot agent and uses them to find obstacles in the path of the robot. Furthermore it has to update a map as the robot moves on, so it also receives information about the current speed of the robot from the robot agent and the current global coordinates from the encoder agent.

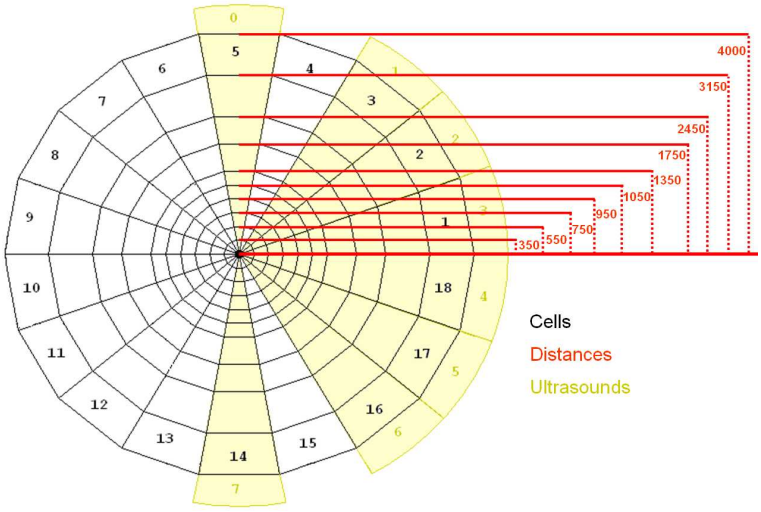


Figure 3.8: Local map of sonar agent.

To create the map, a zone around the robot is divided into cells as shown in Figure 3.8. Cells are obtained by dividing the circle around the robot into 18 circular sectors that represent the ultrasonic sensor visibility zones, and the circular sectors into ten parts representing different distances from the robot. This organization into cells is useful for dealing with noise and fictitious obstacles detected by ultrasonic sensors. If an object is detected in a cell several times, then the probability associated with the cell increases, indicating the

presence of the object. In this way, each cell is given a probability label in relation to whether an obstacle has been detected inside it. Probabilities are increased when an obstacle is sensed and decreased when not. In this way, some ”memory” is introduced into the sensors.

At each sample time, this agent first applies movement to the map (moving objects according to the robot's motion), then updates the sonar information and sets it to the map. After that, and using probabilities, it finds the closest point (the center of the cell where an object has been detected) to the robot, the closest point in front of the robot (this may or may not be the same), and the closest obstacles on both sides of the robot.

After finding the obstacle points, this agent sends them to the avoid and to gothrough agents. To the avoid agent it sends the coordinates of the closest obstacle and the closest obstacle in front of the robot since depending on the situation, one or both points must be maneuvered around. To the gothrough agent, it sends the coordinates of the objects on both sides of the robot. Depending on the situation, the robot will have to navigate between these points. Both agents use the information provided by the sonar agent to calculate the appropriate linear and angular speeds.

This agent can be described in terms of its design pattern, this agent can be described as shown in Figure 3.9.

Agent Design Pattern		Sonar Agent
Internal State		Maintain a consistent local map
Goal:	Configuration	Update local map according to obstacles detected and robot motion.
	Methods	Probabilistic reasoning
Competition		-
Collaboration		Robot agent and encoder agent
Coordination:	Utility Computation	-
	Resource Exchange	-
Helper Methods		

Figure 3.9: Design Pattern of the Sonar Agent

3.4.3 Battery Sensor Agent

This agent's goal is to check from time to time the robot's battery charge level. This is necessary because the robot should recharge batteries when the voltage is around 10V, due to the fact that batteries can be permanently damaged when their level falls below this value. The agent therefore checks the battery level at a given time frequency, which is dynamically increased as the level value drops towards 10V.

This agent receives information from the robot agent, since the latter is the interface with the hardware. In addition, the battery sensor agent not only informs the battery charger agent when requested about the battery level value, but also emits an alarm signal when the value is getting near permanent damage threshold.

This agent can be described in terms of its design pattern as shown in Figure 3.10.

AGENT DESIGN PATTERN		BATTERY SENSOR AGENT
Internal State		Maintain updated the battery charge measure
Goal:	Configuration	Check periodically the battery charge level and emit an alarm if approximating to security threshold.
	Methods	Model-based
Competition		-
Collaboration		Robot agent and battery charger agent
Coordination:	Utility Computation	-
	Resource Exchange	-
Helper Methods		

Figure 3.10: Design Pattern of the Battery Sensor Agent

3.5 BEHAVIORAL AGENTS

The behavior subsystem carries out specific actions, such as avoiding obstacles, going to a point, etc. The information coming from the perception agents is used to react or respond to the changes produced in the robot itself or in the environment.

The basic behavior agents described in this thesis are the goto, the avoid and the gothrough agents, although more agents can be added.

3.5.1 GOTO AGENT

This agent is responsible for driving the robot to the goal position. Given a desired position (x, y) and an orientation θ, and based on the current position and heading, this agent calculates the linear and angular speeds to drive the robot to the target position (goal). It receives the global coordinates of the robot from the encoder agent and the desired points of the trajectory from the mission planning agent or from the battery charger agent. The goto agent sends the linear and angular speeds that are desired to the robot agent.

This agent can be described in terms of its design pattern as shown in Figure 3.11.

AGENT DESIGN PATTERN		GOTO AGENT
Internal State		Maintain motion progress information
Goal:	Configuration	Drive the robot to the goal position with the desired heading
	Methods	Fuzzy Collaborative Control System
Competition		Avoid, gothrough agents for the robot agent
Collaboration		Encoder, mission planning and battery charger agents
Coordination:	Utility Computation	Based on distance to goal position
	Resource Exchange	Fuzzy-based smoothing method
Helper Methods		

Figure 3.11: Design Pattern of the Goto Agent

In order to implement the goal method, a fuzzy-based collaborative control has been chosen instead of developing a complicated, model-based controller.

The starting point is the equation proposed by [49]. In this paper, the author claims that coherent robot behavior can be obtained when several controllers send their set-points directly to motors. The motors themselves act like integrators and the resulting speed depends on the sum of all the control actions divided by the number of controllers, that is, the average sum of all the control actions. The results are good as long as the controllers send the ”right” control actions, but deteriorate if the number of malicious controllers increases.

The idea of implementing the average sum of the control actions to control the robot has been extended by adding the weights corresponding to the relevance of each controller, in accordance with the current context. Thus, the calculation of the desired velocity is defined as a weighted average of the commands provided by the simpler controllers, according to the following formulae:

$$v = \frac{\sum_{i=1}^{n} v_i \cdot \mu_i}{\sum_{i=1}^{n} \mu_i} \qquad \omega = \frac{\sum_{i=1}^{n} \omega_i \cdot \mu_i}{\sum_{i=1}^{n} \mu_i} \tag{3.1}$$

where v_i is the linear velocity and ω_i is the angular velocity desired for the robot by the i controller at any given moment of time, v the real linear velocity of the robot, ω the real angular velocity of the robot, n the number of controllers and μ_i the weights of each controller. These weights satisfy the condition $\sum_{i=1}^{n} \mu_i = 1$.

It is possible, with the weights, to give more or less importance to all of the controllers. In order to introduce this adjustment of to velocity expressed by equation (3.1), a Sugeno functional fuzzy system [37] is proposed, since its output can be a linear function and the implementation of equation (3.1) is straightforward.

Sugeno functional fuzzy systems are described by defining the input variables, the output variables and the rules.

The block diagram of the cooperative controller using the proposed method is depicted in Figure 3.12. The set-point and current data inputs to the controllers are the desired coordinates and orientation and the current coordinates and orientation of the robot. To the block corresponding to the fuzzy systems, inputs are the desired speeds (v and ω) coming from the controllers and the distance to the destination point (d), while outputs are the average sum of the desired speeds (values described by Equation 3.1).

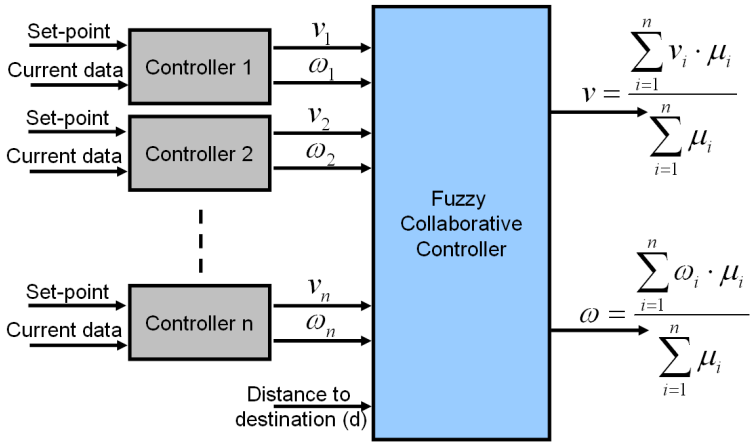


Figure 3.12: Block diagram of cooperative control of the *goto* agent

The different parts of the cooperative controller as shown in this block diagram are de-

scribed in the following subsections.

INPUT VARIABLES

According to Figure 3.12 the collaborative block inputs are the distance (d) and the linear (v) and (ω) speeds of the different controllers. For each input variable several labels are defined (for example, dlabel1), and are modeled by membership functions. Inputs are fuzzified by applying the membership functions to their current values.

d is defined as a function of the distance that the robot has already covered d_{rec} and the distance from the initial position of the robot to the destination point d_{max}, i.e. $d = d_{max} - d_{rec}$. In this case, two possible values for d have been defined: *near* and *far*. The terms *near* and *far* are modeled with fuzzy sets and defined as:

$$\mu_{near}(d) = \left\{ \begin{array}{ll} 1 & d \leq min \\ \frac{(-d+max)}{(max-min)} & min < d < max \\ 0 & d \geq max \end{array} \right\} \qquad (3.2)$$
$$\mu_{far}(d) = 1 - \mu_{near}(d)$$

where min and max are parameters that have been empirically tuned. Figure 3.13 shows these parameters for the current implementation.

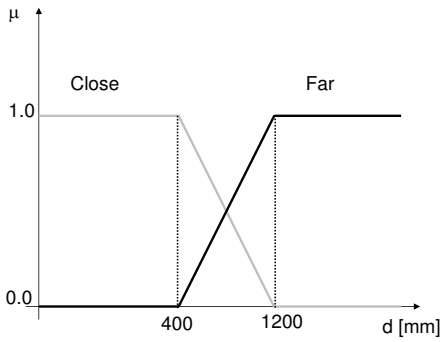


Figure 3.13: Fuzzy Sets

According to this definition, the distance to the destination point can vary in a non-linear way in time, depending on the movement of the robot, causing the fuzzy weights to also be non-linear. Figure 3.14 shows different results for the fuzzy weights described by Equation (3.2) over time. In Figure 3.14-a) a linear case is presented, while Figures 3.14-b) and 3.14-c) reveal the non-linearity of the equation, which also depends on the min and max parameters.

The controller's inputs (linear and angular speeds) have not been fuzzified.

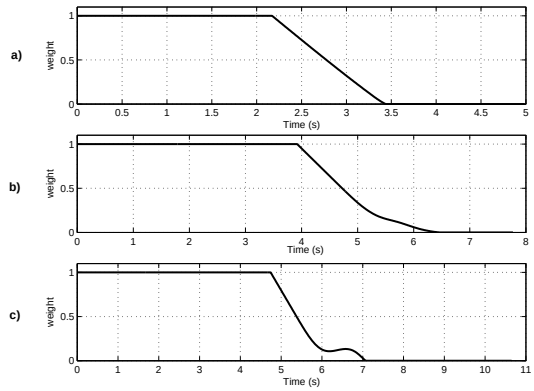


Figure 3.14: Fuzzy weights resulting after applying the near fuzzy set

OUTPUT VARIABLES

The final linear and angular velocities are obtained as output variables (conclusion of the rule). The values of the output variables are determined as linear combinations of the inputs (Sugeno's system with $n+1$ variables). In other words, each value x of the output variable s is computed as follows:

$$u_x(s) = a_{n+1} \cdot v_{n+1} + ... + a_1 \cdot v_1 + a_0 = (a_{n+1}, ..., a_1, a_0) \tag{3.3}$$

where a_i is the coefficient of the input variable v_i and a_0 is a constant. The final output of the system is the sum of the individual outputs averaged by the inputs (that is, using the average function as the defuzzification method).

RULES

In a Sugeno approach, rules have the following structure:

IF **d** IS dlabel1 AND C_1 IS C1label1 AND ... C_N IS CNlabeln
THEN **v** IS vlabel1 AND ω IS ωlabel1

where d,C_1,...,C_N are the input variables and v and ω are the output variables.

3.5.2 AVOID AGENT

This agent is responsible for avoiding the obstacles that could be in the robot's path to its goal. It asks the sonar agent for the closest obstacle points and calculates the suitable linear and angular speeds for avoiding collisions with the objects, regardless of whether they are in front of the robot or on the flanks. In this way, the avoid agent can take better decisions.

This agent receives information from the mission planning agent and the battery charger agent (coordinates of trajectory points), from the encoder agent (current global coordinates), from the robot agent (current linear and angular speeds), and from the sonar agent (current closest obstacle and closest obstacle in front of the robot). It sends the desired linear and angular speeds to the robot agent.

This agent can be described in terms of its design pattern as shown in Figure 3.15.

AGENT DESIGN PATTERN		AVOID AGENT
Internal State		Maintain dodging obstacles progress information
Goal:	Configuration	Avoid obstacles, guaranteeing save motion
	Methods	PID control system
Competition		Goto and Gothrough agents for the robot agent
Collaboration		Sonar, encoder, mission planning, battery charger , robot agents
Coordination:	Utility Computation	Based on time to collision
	Resource Exchange	Fuzzy-based smoothing method
Helper Methods		

Figure 3.15: Design Pattern of the Avoid Agent

In order to calculate speeds, the agent uses three different zones around the objects: the caution zone, the danger zone and the stop zone. When the robot is in the caution zone (see Figure 3.16) the avoid agent calculates a lower linear speed, because it is a long way from the object and there is a low probability of colliding. In the danger zone, the avoid agent changes the linear speed as well as the angular speed, heading the robot out of this zone. If the robot enters the stop zone, the avoid agent stops linear movement and rotates the robot 180 degrees. These zones vary according to the kind of environment the robot is moving in and its current speed. They are narrower if there are many objects and wider if there are few; maximum speeds can be greater in the second case.

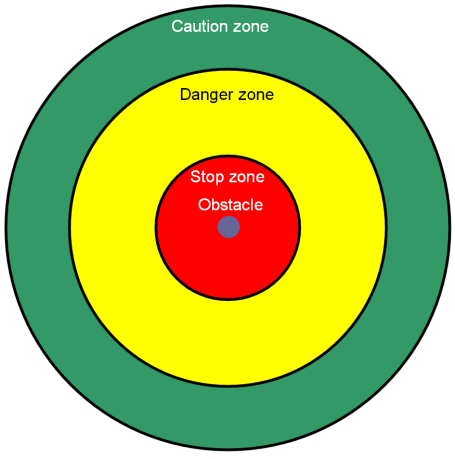


Figure 3.16: Zones around the obstacle point.

3.5.3 GOTHROUGH AGENT

This agent's goal is to drive the robot through narrow places (such as corridors or doors). It asks the sonar agent for the closest obstacles on both side of the robot, determines if the robot is in a narrow place and calculates suitable linear and angular speeds to maintain the robot in the center of the narrow place.

This agent receives information from the mission planning agent or the battery charger agent (desired trajectory points), from the encoder agent (current global coordinates), and from the sonar agent (coordinates of obstacles on both side of the robot). It sends the desired linear and angular speeds to the robot agent.

This agent can be described in terms of its design pattern as shown in Figure 3.17.

AGENT DESIGN PATTERN		GOTHROUGH AGENT
Internal State		Maintain motion in narrow places progress information
Goal:	Configuration	Detect narrow places and drive the robot through them.
	Methods	Model based motion
Competition		Avoid and goto agents for the robot agent
Collaboration		Encoder, mission planning, battery charger and sonar agents
Coordination:	Utility Computation	Based on distance to side obstacles
	Resource Exchange	Fuzzy-based smoothing method
Helper Methods		

Figure 3.17: Design Pattern of the Gothrough Agent

In order to calculate the desired linear and angular speeds, the gothrough agent finds the center point between the obstacles on both side of the robot. Then, and taking into account the final destination, it proposes a straight line between the center point of the obstacles and the destination, and locates the midpoint of this line. This midpoint is projected onto the axis of movement parallel to the objects on the both sides of the robot, defining a point in the middle of the objects. Using this new point, the gothrough agent calculates the appropriate angular and linear speeds. In this way, a new trajectory from the current position of the robot to the destination point is described, making it possible for the robot to remain midway between obstacles in narrow places.

3.6 DELIBERATIVE AGENTS

The deliberative subsystem is composed of agents in charge of carrying out high-level complex tasks which require a certain amount of time.

Four deliberative agents have been defined: the mission planning, the path planning, the localization and the battery charger agents.

3.6.1 MISSION PLANNING AGENT

The mission planning agent's goal is to plan the sequence of tasks based on the information provided by the interface and localization agents, and to ensure that the mission is

achieved. This agent can be described in terms of its design pattern as shown in Figure 3.18.

AGENT DESIGN PATTERN		MISSION PLANNING AGENT
Internal State		Maintain mission progress information
Goal:	Configuration	Achieve the mission
	Methods	Decomposition into tasks (procedural reasoning)
Competition		Battery charger agent for the path planning, goto and gothrough agents
Collaboration		Interface, localization, wakeup and battery charger agents
Coordination:	Utility Computation	Based on mission priority
	Resource Exchange	Trajectory merging
Helper Methods		

Figure 3.18: Design Pattern of the Mission Planning Agent

The mission planning agent receives information from the interface agent about the mission to be achieved, from the battery charger agent to judge whether a mission is possible considering the level of battery charge, and from the localization agent to know the position of the robot on the global map.

The method employed to decompose a mission into tasks is based on a procedural approach similar to PRS [45]. The mission planning agent needs three resources: the path planning agent, the goto agent and the gothrough agent. When the mission planning agent has to deal with a positioning task, it requests a plan from the path planning agent to move from the current to the destination position. The path planning agent is shared by the battery charger agent, which could request a plan to move to a charge battery position at the same time. So both agents, the mission planning agent and the battery charger agent, coordinate to win the path planning resource agent. On the other hand, as stated above, the mission planning agent could request the goto and the gothrough agents to follow a trajectory. These latter agents are shared by the battery charger agent. Both agents decide when a trajectory is feasible, taking into account the remaining energy in the battery.

The mission planning agent can be, at the same time, a multi-agent system as shown in Figure 3.3. in contrast to the rest of the architecture, this multi-agent system is formed by several planner agents and a central coordinator agent (see Figure 3.19). The key agent in the mission planning MAS is the coordinator agent, which will ask the planner agents to create a plan or a sub-plan. The idea is to have several planners that are suitable for different situations (in certain cases some of them can may not provide a solution or the solution provided may not be the best) or that can provide a solution at different times. The coordinator agent will know which planner is appropriate for each situation and will ask it for a plan. It will also ask for re-planning whether there is an unexpected event, or even ask some planners to give a sub-plan for specific parts of the whole plan. The MAS therefore acts as an abstract agent [50] in ARMADiCo architecture.

An example can be used to clarify some of these ideas. Let us consider a surveillance robot that has to move around a building and take pictures of specific rooms. The map of the building is shown in Figure 3.20.

The actions that the robot can perform are: move(from-to), check-alarm and take-photo. Let us suppose that the robot is in Room R2 and must adhere to the following plan:

$\{move(R_2, Hall), move(Hall, R_1), take{-}photo, move(R_1, Hall), move(Hall, R_{11}), take{-}photo, move(R_{11}, Hall)\}$

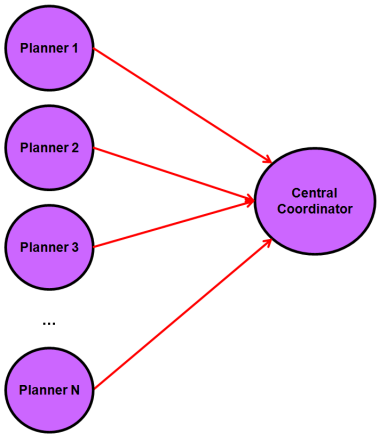


Figure 3.19: Mission Planning Abstract Agent.

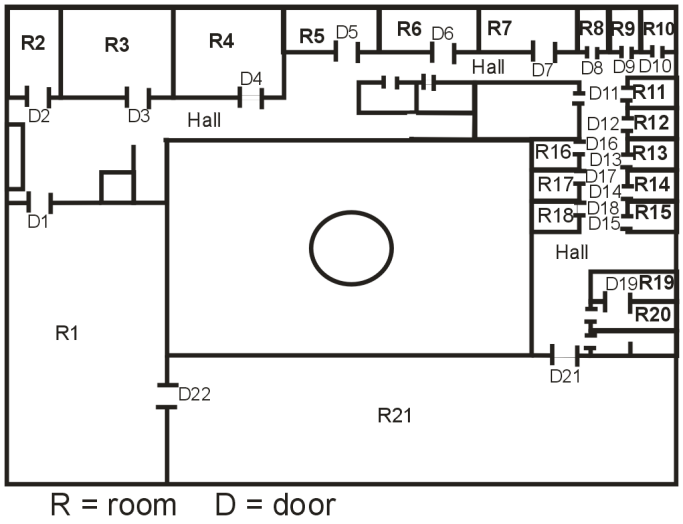


Figure 3.20: Floor plan of the surveillance robot's building.

The robot starts to execute the plan but when it is in Room R_1 taking a picture there is an alarm in Room R_{21}. At this moment, the coordination agent asks the next two planner agents, for a sub-plan that considers going to R_{21} to check the alarm. The answers are:

Planner Agent 1: $\{move(R_1, R_{21}), check - alarm, move(R_{21}, R_1), move(R_1, Hall)\}$

Planner Agent 2: $\{move(R_1, R_{21}), check - alarm, move(R_{21}, Hall)\}$

The proposed sub-plan of Planner Agent 2 may be the best because it will put the robot near the next objective that is Room R_{11}. So, under this belief the coordination agent can modify the original plan to (following a merging procedure):

Final Plan (Coordination Agent): $\{move(R_1, R_{21}), check - alarm, move(R_{21}, Hall), move(Hall, R_{11}), take - photo, move(R_{11}, Hall)\}$

And the robot continues with the original plan.

3.6.2 PATH PLANNING AGENT

The path planning agent has two main goals: the calculation of a trajectory to the goal that is free of non-moving obstacles, and the estimation of the energy consumption of the planned trajectory. The optimal trajectory calculation is obtained in two steps: first with a graph method to obtain a general sequence of destinations (considering only rooms and hallways), and second with a grid method to find the path between two consecutive destinations (considering all the non-moving obstacles). In both methods a search algorithm is used, Dijkstra's algorithm for the first and and the A* search algorithm for the second one. Once the trajectory is determined, an estimation of energy consumption is made based on the cruising speed. Other alternative methods for path planning can also be used, as for example potential fields [108].

This agent can be described in terms of its design pattern as shown in Figure 3.21.

AGENT DESIGN PATTERN		PATH PLANNING AGENT
Internal State		Maintain mission progress information
Goal:	Configuration	Calculation of a free-of-obstacle trajectory and energy consumption
	Methods	Graph and grid search methods
Competition		-
Collaboration		Mission planning, localization and battery charger agents
Coordination:	Utility Computation	-
	Resource Exchange	-
Helper Methods		

Figure 3.21: Design Pattern of the Path Planning Agent

3.6.3 LOCALIZATION AGENT

The localization agent's goal is to locate the robot on the global map. It receives information from the sonar and encoder agents. In order to accomplish the agent's goal, a MonteCarlo technique is used to determine its position and orientation. The localization agent collaborates with the mission planning, the path planning and the battery charger agents by sending them the current position. It also sends information to the encoder agent to prevent wide deviations from the current estimated position. In this way, the encoder agent can correct accumulative errors produced by the encoder's readings.

This agent can be described in terms of the design pattern as shown in Figure 3.22.

3.6.4 BATTERY CHARGER AGENT

This agent, based on the information given by the battery sensor agent, analyzes when it is necessary to go to the recharge point. It sends an alarm to the mission planning agent when the battery level goes under $10.4V$, which indicates that reaching the recharge point is essential, and asks the path planning agent to calculate the best trajectory to this point.

This agent can be described in terms of its design pattern as shown in Figure 3.23.

Agent Design Pattern		Localization Agent
Internal State		Maintain localization progress information
Goal:	Configuration	Locate the robot on the global map
	Methods	Probabilistic MonteCarlo technique
Competition		-
Collaboration		Interface, encoder and sonar agents.
Coordination:	Utility Computation	-
	Resource Exchange	-
Helper Methods		

Figure 3.22: Design Pattern of the Localization Agent

Agent Design Pattern		Battery Charger Agent
Internal State		Maintain energy consumption progress information
Goal:	Configuration	Ask for a trajectory to the recharge point when battery level critical.
	Methods	Model-based method
Competition		Mission planning agent for the path planning, goto and gothrough agents
Collaboration		Battery sensor, path planning, localization agents
Coordination:	Utility Computation	Based on energy level
	Resource Exchange	Trajectory merging
Helper Methods		

Figure 3.23: Design Pattern of the Battery Charger Agent

3.7 Actuator Agents

The actuator subsystem is responsible for the direct use of the robot's various performance motion systems. There can be as many agents as there are actuators or sets of actuators in the robot. For the Pioneer 2 DX robot there is a single actuator agent: the robot agent.

Other agents, such as an arm agent, can be added as and when required.

3.7.1 Robot Agent

The robot agent is the interface between the multi-agent architecture and the robot micro-controller. For this agent, the design pattern can be described as shown in Figure 3.24.

The robot agent represents the real robot in the architecture, in this case MobileRobots commercial Pioneer 2DX. This robot has two driving wheels, motorized with DC motors, fixed at the front of the robot's body, and a free wheel at the back. Each front wheel has an encoder sensor attached. There are also eight ultrasonic sensors placed in a ring at the front of the robot. It is also possible to measure the battery charge during operation.

The robot has a micro-controller and a PC on-board. The micro-controller obtains the sensor readings and controls the motors. Inside, there are two controllers, a linear speed controller and an angular speed controller. Communication with the PC (where the MAS architecture is running) is achieved via RS-232. The ARIA commercial library from Mo-

AGENT DESIGN PATTERN		ROBOT AGENT
Internal State		Maintain communication with real robot progress information
Goal:	Configuration	Be an interface between the MAS architecture and the mobile vehicle.
	Methods	Communication with the robot using ARIA
Competition		-
Collaboration		Avoid, goto and gothrough agents
Coordination:	Utility Computation	-
	Resource Exchange	-
Helper Methods		

Figure 3.24: Design Pattern of the Robot Agent

bileRobots Inc. is used to communicate between the local server (micro-controller) and the agents. The micro-controller acts as a server and only allows one client connection at a time. Thus, it provides information about the different sensors and actuators and computes the position and heading of the robot in local coordinates. Specifically, every 100 ms the robot agent communicates with the robot and obtains its current position, sonar readings and battery charge, and distributes this information to the different sensor agents as required. It also gets the desired angular and linear speed from the behavioral agents and sends them to the micro-controller.

3.8 BACK AGENTS

All the agents necessary to ensure the correct functioning of the multi-agent architecture are identified as back agents. In this group two subsets of agents can be differentiated, one that is accountable for providing the basic services of the multi-agent platform (including agent communication) and the other that is responsible for maintaining the architecture's integrity. They are the directory facilitator agent, the monitor agent, the wakeup agent and the client agent.

3.8.1 DIRECTORY FACILITATOR AGENT

The directory facilitator agent (DF) has been designed in accordance with the Foundation for Intelligent Physical Agents (FIPA-IEEE) standards [38]. It keeps track of the agents currently active in the architecture, their location in the net (host name and port name), the services they provide and require and the resources they need. Furthermore, each agent that registers, tells the DF agent whether it can provide services for several agents at the same time, i.e. whether or not there is mutual exclusion. Each time an agent joins the community, it has to register with the DF agent which then informs the rest of the community about the new agent, the services that it offers and needs and the resources that it uses. Thus, this agent is a key element in the platform in terms of inter-operating issues such as the use of various software programs from different providers.

This agent can be described in terms of its design pattern as shown in Figure 3.25.

The DF agent has a record of all the active agents in the architecture. As agents are registering the DF constructs a table similar to Table 3.1.

Using the information contained in this table, the agents in the community can establish the

Table 3.1: Information about agents in the DF agent

Agent Name	Host Address	Port Number	Provided Services	Requested Services	Resources	Mutex
Robot	120.0.0.1	6240	relative_coordinates relative_heading sonar_heading sonar_readings battery_reading current_linear_speed current_angular_speed	desired_linear_speed desired_angular_speed	None	True
Encoder	120.0.0.1	6235	absolute_coordinates absolute_heading	relative_coordinates relative_heading initial_coordinates initial_heading	None	False
Sonar	120.0.0.1	6236	front_closest_obstacle_distance front_closest_obstacle_heading closest_obstacle_distance closest_obstacle_heading side_closest_obstacles_distances side_closest_obstacles_headings	sonar_heading sonar_readings absolute_coordinates absolute_heading	None	False
Battery Sensor	120.0.0.1	6241	battery_level battery_alarm	battery_reading	None	False
Goto	120.0.0.1	6239	desired_linear_speed desired_angular_speed utility	desired_coordinates desired_heading absolute_coordinates absolute_heading	Robot	True
Avoid	120.0.0.1	6237	desired_linear_speed desired_angular_speed utility	front_closest_obstacle_distance front_closest_obstacle_heading closest_obstacle_distance closest_obstacle_heading desired_coordinates desired_heading absolute_coordinates absolute_heading current_linear_speed current_angular_speed	Robot	True

AGENT DESIGN PATTERN		DIRECTORY FACILITATOR AGENT
Internal State		Maintain community's progress information
Goal:	Configuration	Keep track of agents in the community
	Methods	-
Competition		-
Collaboration		All the agents in the community
Coordination:	Utility Computation	-
	Resource Exchange	-
Helper Methods		

Figure 3.25: Design Pattern of the Directory Facilitator Agent

communication ports in order to collaborate and compete with each other.

3.8.2 MONITOR AGENT

The monitor agent's goal is to police the agents connected in the platform. This agent must be aware when an agent that is essential for the correct functioning of the robot dies, and perform the required steps to restart it or to shut down all the agents (in the case, for example, where the dead agent is the DF) thereby keeping the robot and the environment safe. Also, it must check that communication channels are working properly. This agent receives information from the DF agent, the wake up agent and regularly from the agents in the platform.

This agent can be described in terms of its design pattern as shown in Figure 3.26.

AGENT DESIGN PATTERN		MONITOR AGENT
Internal State		Maintain community's progress information
Goal:	Configuration	Police agents in the community
	Methods	-
Competition		-
Collaboration		All the agents in the community
Coordination:	Utility Computation	-
	Resource Exchange	-
Helper Methods		

Figure 3.26: Design Pattern of the Monitor Agent

3.8.3 WAKEUP AGENT

The wakeup agent is in charge of starting up the minimum configuration of agents required to perform a mission. The wakeup agent receives information from the mission agent saying which kind of mission is going to be carried out. Based on a database, the wakeup agent starts all the agents needed to do it. For example, if the mission is to go to a point, the

wakeup agent will start the robot, the encoder, the sonar, the goto, the gothrough, the avoid and the path planning agents. They all are necessary to safely guide the robot from the initial position to the final position. If the mission is more complicated, then it will also start the localization, the battery charger and the battery sensor agents.

This agent can be described in terms of its design pattern as shown in Figure 3.27.

AGENT DESIGN PATTERN		MONITOR AGENT
Internal State		Maintain community's progress information
Goal:	Configuration	Start the minimun configuration of agents in the community to perform a mission
	Methods	-
Competition		-
Collaboration		All the agents in the community
Coordination:	Utility Computation	-
	Resource Exchange	-
Helper Methods		

Figure 3.27: Design Pattern of the WakeUp Agent

3.9 SOCIAL AGENTS

Social agents[2] are meant to communicate with other agents (human or software) of other communities. They are the links between communities. In ARMADiCo the Interface agent has been defined as a social agent.

3.9.1 INTERFACE AGENT

The interface agent's goal is to interact with a human operator or external software agent, but it also visualizes the progress of the mission while the robot is executing it. It can follow up the robot's position, trajectories, messages among agents, obstacle information, etc. It can show a local map, with the obstacles detected by the robot and the global map with trajectories and the available readings of the different sensors. This agent can also save all this information in a file.

This agent can be described in terms of its design pattern as shown in Figure 3.28.

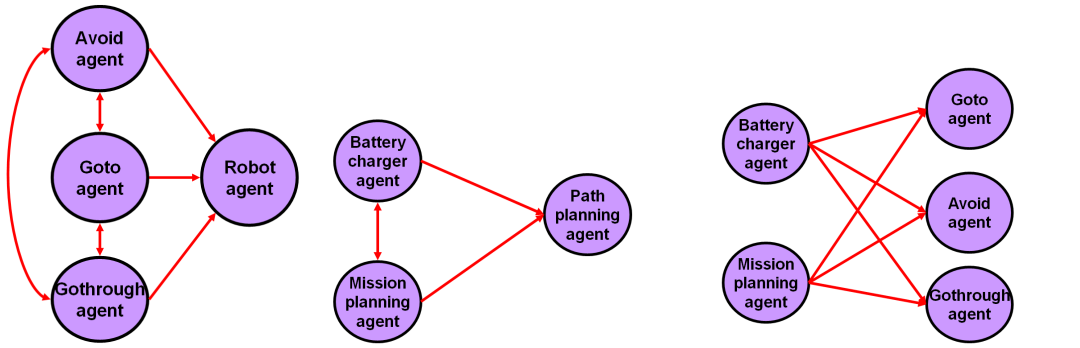
3.10 COORDINATION

Coordination among agents is necessary when there are several agents trying to use the same resource at a given time. In ARMADiCo, these conflicts can arise among the avoid, the goto and the gothrough agents when trying to send conflicting actions to the robot agent and between the battery charger and the mission planning agents when demanding a trajectory from the path planning agent or when sending the trajectory points to the goto and the gothrough agents, as shown in Figure 3.29. Here, circles represent the agents, and arrows the communication flow between two agents.

[2]Note that social agents (in the architecture) differ from the social intelligence of the agent (inside the agent).

AGENT DESIGN PATTERN		MONITOR AGENT
Internal State		Maintain mission progress and agents information
Goal:	Configuration	Keep track of all the events in the control architecture and to communicate with other agents out of the community
	Methods	-
Competition		-
Collaboration		All the agents in the community
Coordination:	Utility Computation	-
	Resource Exchange	-
Helper Methods		

Figure 3.28: Design Pattern of the Interface Agent



a) Behavioral agents *b) Deliberative agents* *c) Deliberative and reactive agents*

Figure 3.29: Possible conflicts in ARMADiCo.

In ARMADiCo each agent computes a normalized utility value (between [0,1]) only known by the agent itself. In order to solve conflicting decisions, the utility value is used to obtain control over the resources. This utility value is sent to the other agents in conflict in order to decide on the best action to perform. Thus, the agent that has the highest utility value makes the decision. This agent gets control over the activity giving rise to the conflict. For example, suppose that the goto agent has a utility value of 0.5, the gothrough agent of 0.3 and the avoid agent of 0.7, with 0.7 being the highest value. The avoid agent takes control of the situation, and is the only one that sends speeds to the robot agent.

To reduce communication among agents using this decentralized approach, the agent that has control broadcasts its utility value. If there is no response, meaning that it has the highest value, the agent uses the resource. Conversely, if there is an agent with a higher utility value, then it informs all the agents of this value, indicating that it is going to use the resource. In this way, the communication process is reduced and centralization of coordination is avoided.

Each agent can have its own method for each resource and this determines its social intelligence (see Section 3.3.2). For the sake of extension, the scope of the current ARMADiCo implementation has been limited to the methods employed by the behavioral agents. They are described below.

Figure 3.30 shows the coordination state diagram for the conflicting action depicted in Figure 3.29-a). Circles indicate the states and the arrows are messages between the agents. The parameters of the message refers to the emitter agent and to the receiver agent; for ex-

ample, send_utility(goto,avoid) means that the goto agent sends its utility value to the avoid agent. The goto agent changes from the initial state to state number 1 when it receives a message from the mission planning agent indicating the desired position. It remains in state 1 until the encoder agent sends the current position. At this moment it changes to state number 2. Here the goto agent computes the desired angular and linear speeds and the utility value and sends the later to the other agents (conflicting agents), and changes its state to number 3. At this state, it continuously computes the desired speeds based on the information given by the encoder agent, and computes its utility value. Each time, it sends the utility value to the conflicting agents (in this case, the avoid and the gothrough agents) and waits a short time for an answer. If there is no message, the cycle continues and the agent sends the desired speeds to the robot agent. If a higher utility value is received, the goto agent changes its state to number 4. Based on the information received from the encoder agent, it computes the desired speeds and the utility, but in this new state it does not send the utility to the other agents in conflict. It remains listening to the agent with the highest utility, which has taken control of the robot. When the utility is the highest, the state changes to state number 3. Changes from state 3 to 4 and vice versa are performed until at a given moment the desired position is reached. This can only happen from state 3, when the goto agent has control of the robot. Then a null speed message is sent to the robot, ending the whole process.

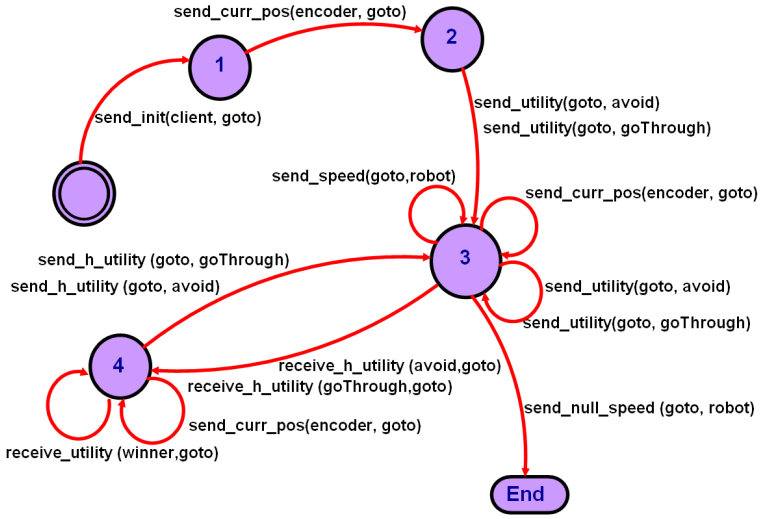


Figure 3.30: Coordination state diagram (goto-avoid-gothrough).

Thus, in this coordination mechanism two main issues need to be distinguished: the winner determination method (or how utilities are computed) and the resource exchange method (or how the winner takes control of the resource).

3.10.1 Winner Determination Method

The behavioral agents have the same method for calculating the utility value, represented by the following expression:

$$u = \begin{cases} p_{max} & x < u_L \\ \alpha * x + \beta & u_L \leq x \leq u_H \\ p_{min} & u_H < x \end{cases} \tag{3.4}$$

being:

$$\alpha = \frac{p_{max} - p_{min}}{u_L - u_H} \quad \beta = \frac{u_L * p_{min} - u_H * p_{max}}{u_L - u_H} \tag{3.5}$$

where p_{max} and p_{min} are the maximum and minimum values, respectively, of u ($p_{min} \leq u \leq p_{max}$); and u_L and u_H represent distance thresholds, which are found empirically.

Figure 3.31 shows the generic representation of the utility function. Parameters x, p_{max}, p_{min} , u_L and u_H are characteristics of each agent.

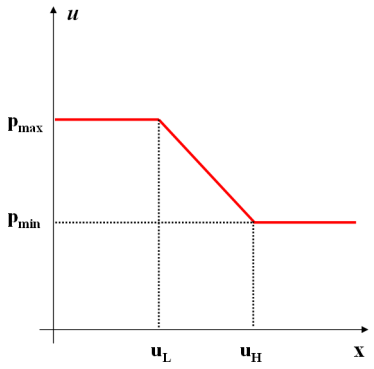


Figure 3.31: Generic form of the utility function.

Once the values of the utility are calculated, the agent controlling the robot sends a message to the other agents in conflict to inform them of the value of its utility. If this value is greater than that calculated by the receiving agents, the other agents do not respond to the message, and the initial agent continues controlling the robot. If, on the other hand, the utility of one of the receiving agents is greater, it then answers the message sure that the other agents know that it will take control of the robot. This last agent will have control until its utility is smaller than that of the other agents.

The way of selecting the behavior that takes control is contained in the arbitration methods defined by [103], but the decision about which agent takes control is made by the agent itself, to prevent only one agent from centralizing the entire architecture.

GOTO AGENT

In the case of the *goto* agent the parameter x is defined as the distance (d_l) that remains to reach the destination location. Also, $p_{max} = 1$, $p_{min} = 0.6$, $u_L = 150mm$ and $u_H = 500mm$. Thus, equation 3.4 becomes:

$$uGoto = \begin{cases} 1 & d_l < 150mm \\ -0.00114 * d_l + 1.1714 & 150mm \leq d_l \leq 500mm \\ 0.6 & 500mm < d_l \end{cases} \tag{3.6}$$

And the form of the utility function is shown in Figure 3.32.

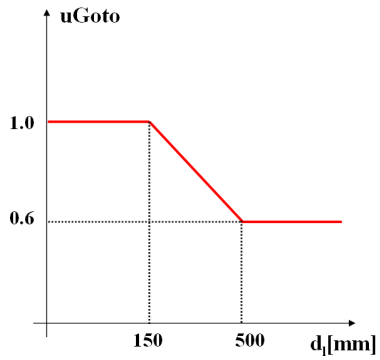


Figure 3.32: Utility function for the goto agent.

GOTHROUGH AGENT

The gothrough agent calculates the distance between both obstacles and divides it by 2. Then, it uses this distance (d), as the parameter x, to calculate the utility value. The more separate the obstacles are, the less value the utility has. Also, $p_{max} = 1$, $p_{min} = 0$, $u_L = 500mm$ and $u_H = 1200mm$. Thus, equation 3.4 becomes:

$$uGothrough = \begin{cases} 1 & d < 500mm \\ -0.00143 * d + 1.7143 & 500mm \leq d \leq 1200mm \\ 0 & 1200mm < d \end{cases} \tag{3.7}$$

And the form of the utility function is shown in Figure 3.33.

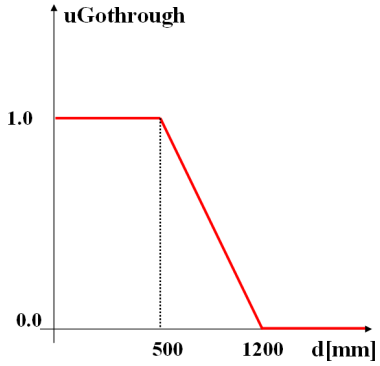


Figure 3.33: Utility function for the gothrough agent.

AVOID AGENT

For its part, the avoid agent has two parameters x on which the calculation of the utility value is based: the distance that remains before a collision with the obstacle (d_o) and the orientation of the robot with respect to the obstacle (h_o) (see Figure 3.34).

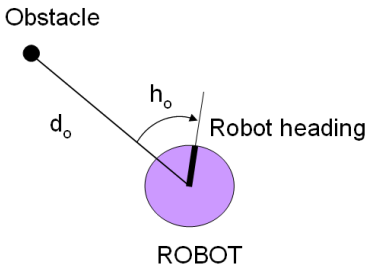


Figure 3.34: Parameters involved in the calculation of the utility of the *avoid agent*

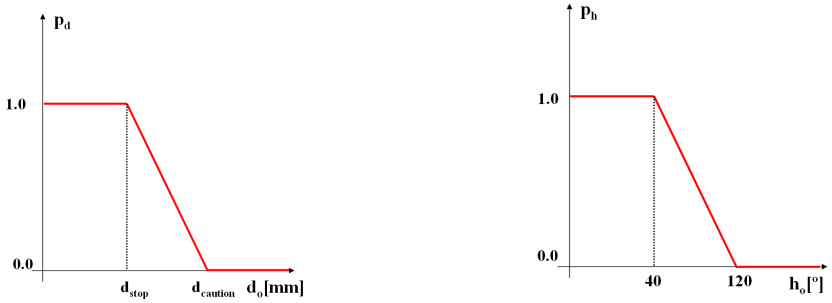
The equation to calculate the utility value is, then:

$$uAvoid = min(p_h, p_d) \tag{3.8}$$

with:

$$\begin{aligned} p_h &= \alpha_1 * h_o + \beta_1 \\ p_d &= \alpha_2 * d_o + \beta_2 \end{aligned} \tag{3.9}$$

and α_i and β_i defined by Equation (3.5), in this case $p_{max} = 1$ and $p_{min} = 0$ being the maximum and minimum values of p_h ($p_{hmin} \leq p_h \leq p_{hmax}$) and those of p_d ($p_{dmin} \leq p_d \leq p_{dmax}$), respectively. In the case of the heading $u_L = 40°$ and $u_H = 120°$ and in the case of the distance to collision, $u_L = d_{stop}$ and $u_H = d_{caution}$. These distances are the zones defined to calculate the linear and angular speeds (refer to Figure 3.16). It should be recalled that they are proportional to the speed of the robot, so, in the calculation of p_d what is really being considered is the time that would remain before the robot collided with the obstacle if it was going directly toward it. Applying Equation (3.8), the time until collision, assuming that the robot is headed directly towards the obstacle, is taken into account, as is the orientation.



a) Function of the distance to the obstacle *b) Function of the heading to the obstacle*

Figure 3.35: Utility function for the avoid agent.

3.10.2 Fuzzy-based Method for Resource Exchange

In relation to robot behavior, changes of commands have been studied in the literature at the control level [49, 55, 113]. Most of the approaches are based on the fusion or arbitration

of control, which requires knowledge of the commands given by several controllers at each moment.

In ARMADiCo architecture, when an agent obtains the opportunity to use a robot shared resource after coordination, the agent has to proceed with the resource usage taking into account its impact on the physical world in a manner similar to control fusion. To do so, a method based on the information used in the coordination process is proposed.

Let a_w be the agent that wins the resource, and let a_l (which has been using the resource until now) be the agent that loses it. For example, if the shared resource is the robot agent, a_w could be the goto agent, and a_l could be the avoid agent.

Let a_r be the shared resource, or agent that owns a shared robot resource. The resource is characterized by n parameters that configure the possible actions requested of the resource (for example, robot commands). Let $a_{w_1}, \ldots, a_{w_n}$ then be the actions requested by the winning agent and $a_{l_1}, \ldots, a_{l_n}$ those of the losing agent.

First of all, an available time window frame t_f to perform the resource action exchange must be determined, i.e. how much time there is available to change from the current action of the losing agent to the current action of the winning agent. This time depends on the criticality of the robot state, i.e. the change to:

- Non critical situations
- Critical situations

How t_f is computed is specific to each agent.

Then, a progressive change from a_{l_i} to a_{w_i} is performed according to the following expressions:

$$\frac{\delta_w(t, u_w) * a_{w_i} + \delta_l(t) * a_{l_i}}{\delta_w(t, u_w) + \delta_l(t)} \tag{3.10}$$

where $\delta_w(t, u_w)$ is the contribution of a_{w_i} in time t, and $\delta_l(t)$ is the contribution of a_{l_i} and u_w and u_l the utilities. Time t is measured in robot cycles. So after the first cycle, since the winning agent obtains the resource, t_1, the contribution of δ_l should be higher than that of δ_w. As the cycles go on, the value of δ_w gains importance; at a given moment, t_f, the action value should be that of the winning agent, so δ_w should have the highest value (1) and δ_l the lowest, (0).

On the basis of that desired behavior, δ_l has been defined as a linear function in [0, 1], as follows:

$$\delta_l(t) = \frac{u_l}{t_f} * (t_f - t) \tag{3.11}$$

where t_f is the ending change cycle defined above and u_l the utility value of the losing agent. Figure 3.36 shows the plot of the function.

Analogously, δ_w varies in a complementary way to δ_l but it should be taken into account that information regarding the winning utility is updated in each cycle as follows:

$$\delta_w(t, u_w) = \frac{u_w}{t_f} * t \tag{3.12}$$

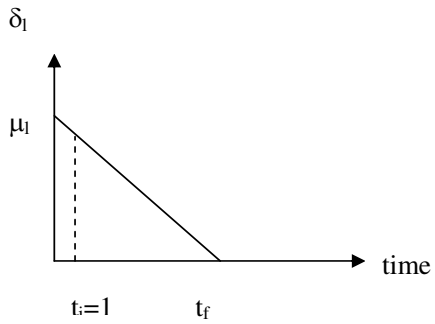


Figure 3.36: Delta for the ”losingr” agents.

Figure 3.37 shows the plot of the δ_w function.

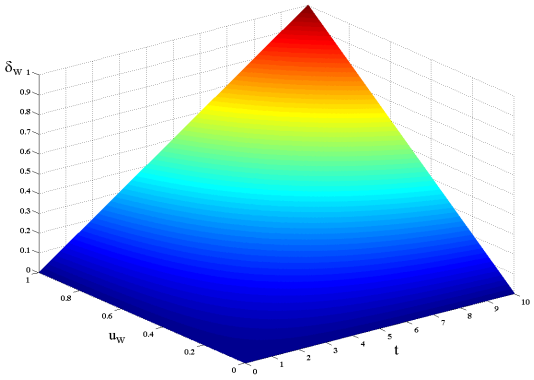


Figure 3.37: Delta for the ”winning” agents.

Note that the resource exchange is performed by the winning agent, which knows all the information regarding the utilities of the losing agent, as well as the actions. However, the information about the losing agent is a ”snapshot” of the losing agent values at the moment that the resource exchange is performed, while the values of the winning agent can be updated in each robot cycle.

The summary of the coordination deploy algorithm is shown in Figure 3.38. Note that step 2.2 is simplified, but involves agent utility computation, coordination information exchange and others.

TIME WINDOW FRAME TO NON-CRITICAL SITUATIONS

Coordination exchange to non-critical situations can happen, for example, when the winner is the goto or the gothrough agents. In both cases a Sugeno fuzzy system is used to determine the time window frame t_f.

The time window frame t_f depends on how different the actions are between the losing and the winning ones. To measure this difference, Tchebychev distance applied to the most

1. Find the time window frame (t_f)
2. Set i=1
3. While u_w is the resource winner and $i < t_f$ do
   {
   3.1 Compute current action parameters
   $$a_c = \frac{\delta_w(t,u_w) * a_{w_i} + \delta_l(t) * a_{l_i}}{\delta_w(t,u_w) + \delta_l(t)}$$
   3.2 Execute a_c
   3.3 Next i
   }

Figure 3.38: Coordination deployment algorithm.

critical action parameters is used, as follows:

$$max_{criticalparameter(1,\ldots,n)}(|a_{w_i} - a_{l_i}|) \tag{3.13}$$

The $criticalparameter(1,\ldots,n)$ function is defined for each shared resource, and returns the set of the resource's critical parameters. For example, in the case of the robot agent, the critical parameter is the linear velocity.

So if this distance is large, the exchange period should be longer, while if the distance is short, the period of change should be close to 0. The concepts of large and short can be easily modeled by following fuzzy variables, and then computing the time window frame t_f according to a fuzzy system.

Specifically, a Sugeno fuzzy system has been followed. In such a system, the input variables, the output variables and the rules are distinguished. There is a single input variable "diff". This variable represents the difference between the action parameters based on the distance function defined in Equation 3.13.

Four different values are defined for the fuzzy variable: equal, small, medium, and big. For a given numerical value of diff, a Gaussian membership function is linked to each value. Figure 3.39 shows the functions used for each fuzzy value.

There is also a single output variable, which represents the length of the time window frame, t_f. Each value of t_f in [0,10] represents the number of cycles required to change from the current action parameters to the new ones. Four different values have been defined for t_f: *nul*, *short*, *medium*, *long*. In a Sugeno system, each value of a output variable is a linear combination of the input variables. That is:

$$t_{fx} = c_1 * \mathtt{diff} + c_2 \tag{3.14}$$

In this case, c_2 is constant and equal to $c_2 = 0$, and the following c_1 coefficients for each t_f value have been defined:

$$t_{fnul} = 0 * \mathtt{diff} \tag{3.15}$$

$$t_{fshort} = 3 * \mathtt{diff} \tag{3.16}$$

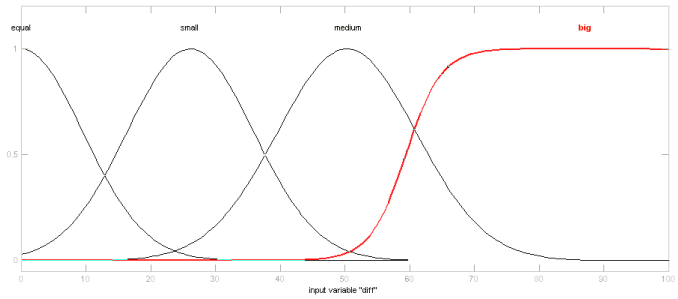


Figure 3.39: Input fuzzy values of the diff variable.

$$t_{fmedium} = 5 * \texttt{diff} \tag{3.17}$$
$$t_{flong} = 10 * \texttt{diff} \tag{3.18}$$
$$\tag{3.19}$$

In relation to the rules, four possible ones have been considered, according to the possible input and output values. These are as follows:

R1: If `diff` is equal then t_f is nul
R2: If `diff` is small then t_f is short
R3: If `diff` is medium then t_f is medium
R4: If `diff` is big then t_f is long

Finally, the output value of t_f is defuzzified by using the weighted mean.

TIME WINDOW FRAME TO CRITICAL SITUATIONS

The coordination exchange to a critical situation proceeds when, for example, the winning agent is the avoid agent, and the robot agent is the shared resource. In this case, the avoid agent has detected a dangerous situation and the time required to react to it, t_c is critical. Therefore, $t_f = t_c$. Note that t_c is known to the winning agent, which applies the coordination deployment method.

The avoid agent calculates t_c as a function of the expected time to collision.

3.11 CONCLUSIONS

In this chapter ARMADiCo architecture has been described. It has been introduced as a new way of deploying robot architectures by exploiting distributed coordination mechanisms of multi-agent systems.

First, the general blocks that form the hybrid architecture as well as the agents that form these blocks have been explained. Next, in order to help to develop new agents, an agent's design pattern has been defined. This design pattern follows a modular structure to integrate all the basic components of an agent. Among these components it is worth highlighting the goal module and the coordination module.

The goal module represents the individual intelligence of the agent and is intended to achieve the agent's goals. Several artificial intelligence tools have been used to implement this module, according to whether the agents belong to the social, the deliberative, the behavioral, the perceptional or the actuator components described in Figure 3.1. Thus, search methods, fuzzy logic, PID, probabilistic approaches and other methods have been defined in heterogeneous agents.

The coordination module represents the social intelligence inside the agent and allows interaction with the other agents in ARMADiCo, in order to cope with the distributed coordination mechanism. This module has been separated into two parts, the winner determination method and the resource exchange method. The winner determination method implements a utility function that allows the agent to know whether it has the control of the resource or not, and uses fuzzy logic to calculate utility values.

On the other hand, the resource exchange method is implemented as a way of dealing with physically grounded resources. This method is the same for the behavioral agents and is based on fuzzy logic. In this case, it is based on a time frame window, the value of which depends on whether the change of control of the resource is critical or not. In the former case, the time window frame is a fixed value and in the latter case it is calculated using a Sugeno approach.

The architecture has been presented and illustrated with several examples. How it was tested is shown in the next chapter.

CHAPTER 4

Experimental Setup

I saw them from the beginning, when the poor robots couldn't speak, to the end, when they stand between mankind and destruction.

From *I, Robot* by Isaac Asimov

In this chapter the multi-agent methodology applied in the designing of ARMADiCo, the multi-agent platform used in its implementation and the nature of the mobile robot's most important characteristics are explained. A dynamic model of the robot and the simulation softwares that have been developed or used are then presented. At the end, a description is given of the goto agent's collaborative controllers.

4.1 INTRODUCTION

Testing a robot architecture is not easy, since it involves a large number of steps from the design stage through to robust movement in the real world. The process needs to be carefully planned so that the usefulness of the software that has been developed and the repeatability of results can be assured. In the first place, a multi-agent development methodology has to be used to help the designer to debug and test the different steps that transform the specifications into the system that is implemented. Using a multi-agent development methodology ensures that a complete and consistent model of the multi-agent architecture is obtained. Some multi-agent development methodologies have tools that make it possible to get the code of the agents in a programming language, such as Java.

Secondly, and related to the coding of agents, an adequate operating system and programming language must be chosen. These will be closely related to the intended application of the multi-agent system. In the case of real processes, like the mobile robot, an operating system must have certain specific characteristics, such as having control over the executing processes or guaranteeing constant time intervals. For this particular application, operating systems that allow working with real-time constraints are desirable. At the beginning, the QNX operating system was used to develop a control architecture for the robot, but its cost and programming difficulty forced a change to Linux (with the real-time module). The programming language must also support real-time coding, so the C++ programming language was selected, above all because it is compatible with the libraries provided by MobileRobots Inc. for controlling their robots.

Finally, the architecture has to ne adapted to the available hardware. It is therefore useful to have simulation software for a particular mobile robot in order to test the architecture in

a way that is safe for both the robot and for the environment. Thus a good mobile platform model for the correct design of the control laws of the different controllers in the agents is required.

4.2 Multi-agent Development Methodology

At the beginning of this work, several methodologies were analyzed and the Multi-agent Systems Engineering (MaSE) methodology was selected from among them. The two most important reasons for choosing MaSE were:

- *Graphical tool:* the agentTool is a graphical interface that allows the MaSE methodology to be followed in an intuitive way. In addition, agentTool allows different phases of the methodology to be modified when the results are not the expected ones.
- *Based on UML:* there are several works ([128, 67]) in the field of distributed control that use Unified Modeling Language (UML) to analyze the design of complex control systems. This is a significant feature in view of the fact that the architecture must control a real mobile robot.

MaSE methodology has two differentiated parts: analysis and design, as shown in Figure 4.1. The first has three phases: capturing goals, applying use cases and refining roles. The design part, on the other hand, has four phases: creating agent classes, constructing conversations, assembling agent classes and system design. These steps can be applied iteratively to ensure each model is complete and consistent.

Some of the results of the analysis phases were presented in Chapter 3 and the results of applying the capturing goals phase, which is for identifying and structuring goals, are shown in Figure 3.4. The next step, applying use cases, is meant to be used for translating goals into roles (abstract description of an agent's goal) and associated tasks. It is divided into two sub-phases: the use cases, which are a narrative description of a sequence of events that define the desired system behavior, and the sequence diagrams, which show the sequence of events among multiple roles (and define the minimum communication among the roles). Three examples of use cases were presented at the end of Section 3.2 and in Figure 4.2 an example of a sequence diagram for use case 2 is shown.

Next, the refining roles step is for the purpose of identifying all the necessary roles and developing tasks that define role behavior and communication patterns. The following step is creating agent classes where the agents are identified from the roles. The resulting agent class for ARMADiCo is shown in Figure 3.5. The step of constructing conversations that follows is often performed in parallel with the succeeding step of assembling agents. The former defines a coordination protocol between two agents and in the latter the internals of the agent's classes are created. Finally the system deployment defines the configuration of the current system to be implemented.

MaSE has helped to design ARMADiCo and test if the design is coherent.

4.3 Multi-agent Platform

As the multi-agent architecture must control a real robot, it has to manage real-time issues. At the beginning of this work, several commercial multi-agent platforms were analyzed and from among them Open Agent Architecture (OAA) was selected. After developing some agents and interacting with the platform, it was found that the facilitator agent centralized

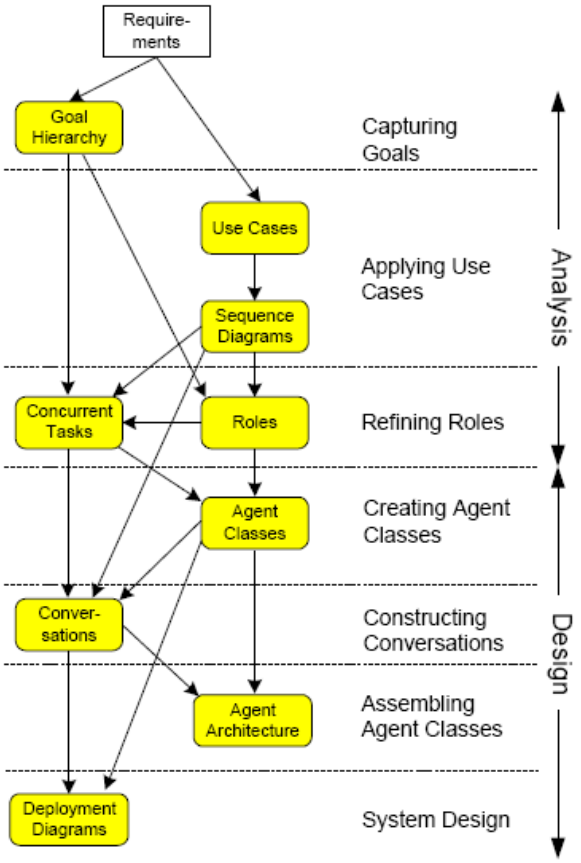


Figure 4.1: MaSE methodology (extracted from [35]).

all of the architecture and thereby impeded some of the desirable characteristics of ARMADiCo architecture. So, an ad-hoc platform was developed. This platform is programmed in C++, it can run on Linux and it is possible to use real-time commands. The important characteristics of both platforms are explained below, highlighting the current implementation of the multi-agent platform.

4.3.1 Open Agent Architecture

Open Agent Architecture (OAA) was developed by the SRI team [77, 78]. The advantages of this platform are that agents can be programmed in C++, and that it can run on Linux. These characteristics allow the use of the real-time Linux module to program reactive agents and to re-use some of the existing code. Another important consideration that was taken into account was that OAA was integrated with ARIA and some commercial libraries that come with the pioneer robot [51] and are used to program the reactive agents.

OAA has particular agents to guarantee the correct functioning of the platform. One worth mentioning is the facilitator agent, which provides the agent community with a number of services for routing and delegating tasks and information among agents. The role of this agent is important because it is where, upon connection, each agent registers its functional capabilities and the specifications of its public data. When a request is sent to the agent community specifying at a high level the description of the task along with optional

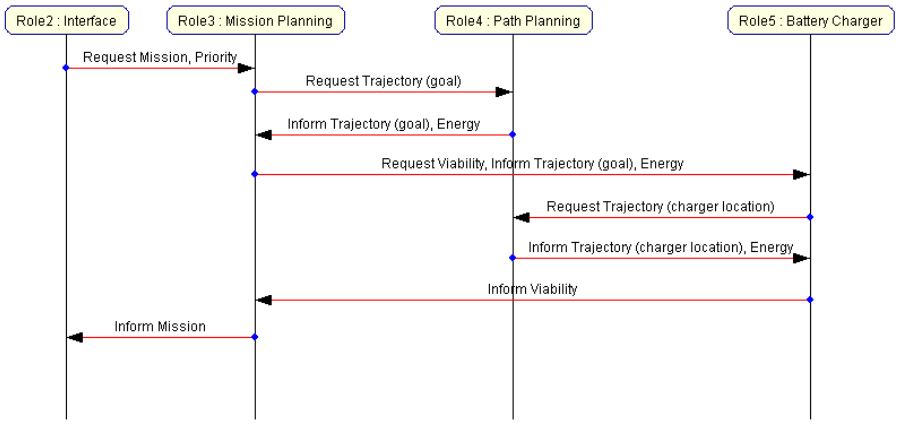


Figure 4.2: Sequence Diagram of Use Case 2.

constraints and advice on how the task should be resolved, the facilitator agent distributes subparts of the task among agents and coordinates their global activity.

Figure 4.3 shows the structure of OAA while highlighting the facilitator agent and the Inter-agent Communication Language (ICL). This language is the interface, communication, and task coordination language shared by all agents, regardless of what platform they run on or what computer language they are programmed in. OAA agents employ ICL to perform queries, execute actions, exchange information, set triggers, and manipulate data in the agent community.

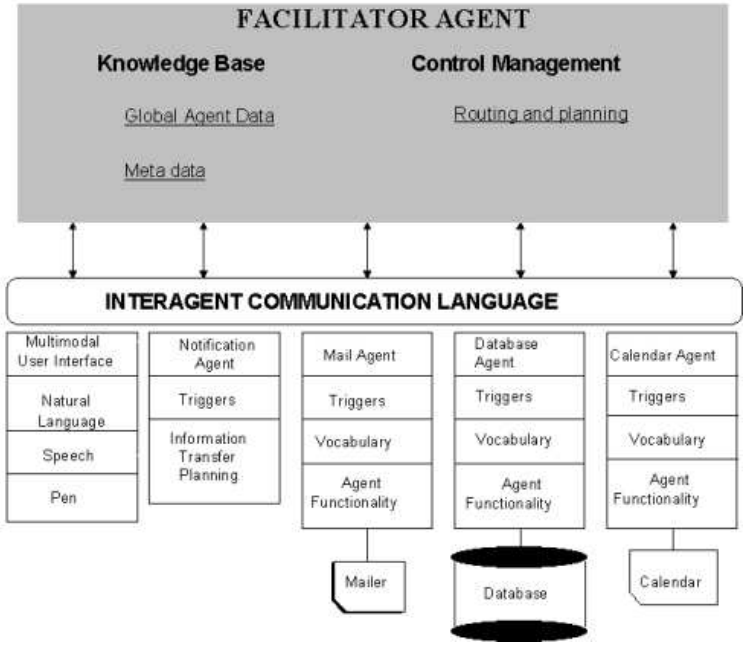


Figure 4.3: OAA structure.

Despite these advantages and this versatility, all interactions between agents, even simple pass message, have to be executed through the facilitator agent. This agent therefore becomes a bottleneck in the architecture, preventing the desired characteristic for ARMADiCo of being completely distributed. Because of this problem, the ad-hoc platform was developed.

4.3.2 ARMADiCo Multi-agent Platform

According to the desired characteristics of the multi-agent architecture described in Table 2.1, the platform must support several functionalities and properties, as for example, executing in real-time if necessary and at different rates, allowing communication among agents without using a central agent to connect them (like the facilitator agent in OAA), allowing agents to execute in different computers and allowing different programming languages for the agents. Thus, the operating system chosen to develop ARMADiCo was Linux and the programming language was C++. It is possible to execute the agents in Windows (previous recompilation of the C++ code). The communication among agents is via TCP and is full duplex (see Figure 4.4). In order to establish the communication channels, each agent has a server to which agents that need to send information will connect, and each agent has a list of clients to which to send information. When connecting, agents must specify the IP address and the port through which messages will be sent.

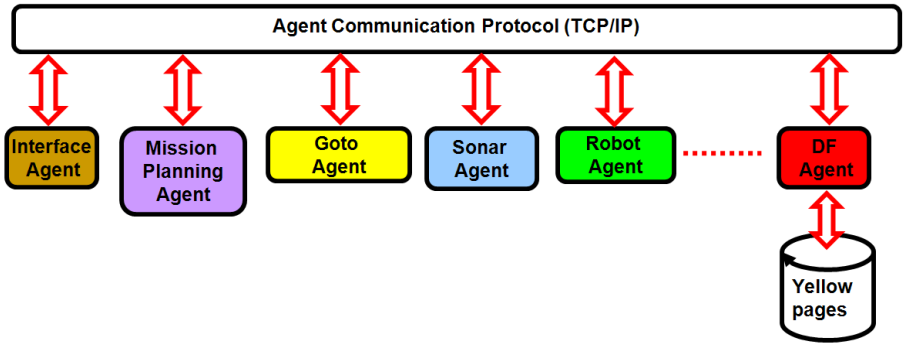


Figure 4.4: ARMADiCo structure.

In order to implement the multi-agent platform, FIPA recommendations [38] have been followed, including some of the services proposed by this standard. A basic agent has also been defined. This agent implements the helper methods as well as some basic functions of the different modules defined in the agent design pattern. To create a new agent, an instantiation of the basic agent must be performed, thereby avoiding having to re-code the basic functionalities (if programming in C++).

Following the agent design pattern, there are several internal variables that are used to keep the internal state of the agent updated (internal state module). Also, there are dynamic lists of agents to cooperate with (collaboration module), of agents to compete with and list of shared resources (competition module). These lists are filled up as agents join the community (after registering with the directory facilitator agent). With respect to the coordination module, the utility computation and the resource exchange method are functions that can be overwritten if necessary. They have a default calculation presented in Section 3.10. The goal module is a function, too. Unlike the coordination module, this function must be fulfilled for each agent with the specific technique to achieve goals. Finally, the helper methods module is distributed among several functions that mostly have to be invoked by passing some configuration parameters.

When creating a new agent with the aim of automating some of the processes to be carried out, the following considerations must be taken into account:

- *Definition of the Requested Services, Provided Services and Shared Resources:* this must be done when the agent is instantiated. Three lists must be created when calling the constructor of the agent. These lists will be used to automatically create the connections with the servers of the agents who request the services provided by

the agent; to create the clients to the server of the new agent from the agents that provides the services the new agent needs; and to indicate the agents that share resources with the new one and that need to be considered when solving conflicts.

- *Agent Name:* a name unique to the agent must be assigned at instantiation. This name, along with the IP address and the port number, will unambiguously identify each agent in the community.
- *Register:* a new agent must register first to the directory facilitator (DF) agent when it is instantiated, and then to the agents that can share resources or that require the services the new agent provides. This latter step is taken while agents are joining the community.
- *Deregister:* this must be done when the agent leaves the community and means closing the TCP connection with all the agents and deregistering from the DF agent.
- *Sending Messages:* to request information or to provide information about something. Messages follow a specific protocol. This protocol is already implemented and the programmer only has to call a function with the appropriate parameters.
- *Receiving Messages:* messages are decoded automatically and the information they have is stored in the internal variables of the agent.

The messages sent by the agents follow FIPA standards. The parameters needed are the identification of the addressee and of the sender, the identification of the information requested or sent, the data (information) and the time the message was sent. The identification of the information requested or sent is predefined,for example: SONREAD meaning sonar readings, POSINI for initial position, and so on. This list can be easily augmented when adding new agents that require different information.

Some of the basic services that the platform needs according to FIPA standards are implemented in the back agents. Thus, the yellow pages are implemented in the DF agent (for register, deregister and modify functions), while the agent management service (AMS) is spread into the wakeup and monitor agents (for register, deregister, modify, create, suspend and execute an agent function).

4.4 THE PIONEER 2DX

The MobileRobots Inc. Pioneer 2DX of [109] is a three-wheeled mobile robot (see Figure 4.5) $44\ cm$ long, $33\ cm$ wide and $22\ cm$ tall. It weighs $16.55\ kg$. Its body is of strong aluminum. At the rear it has 3 $12\ V$ batteries that allow it to run continuously for 8 to 10 hours. At the front, it has two driving wheels with the same axis of rotation, motorized with DC motors. A free wheel is connected to the platform by a rigid structure and can rotate around its vertical axis. The maximum speeds that the robot can reach are $1.6\ \frac{m}{s}$ linear and $300\ \frac{^\circ}{s}$ angular.

The robot has a sonar array of eight ultrasound sensors at the front to detect obstacles, two encoders to determine the position of the wheels, and a battery level sensor to check the charge of the batteries. The sonar array also allows the position of an obstacle to be determined. There are two ultrasound sensors on each side of the robot (at 90°) and the other six are placed at 20-degree intervals, as shown in Figure 4.6. The sonar acquisition rate is $25\ Hz$ and sensitivity ranges from $10\ cm$ to $5\ m$.

The two high-resolution optical quadrature shaft encoders are placed one in each driving wheel. They provide 500 ticks per revolution and 76 pulses for each millimeter of rotation of the wheel (wheel diameter is $165\ mm$ and motor gear ratio is $19.7 : 1$).

Figure 4.5: Pioneer 2DX Robot.

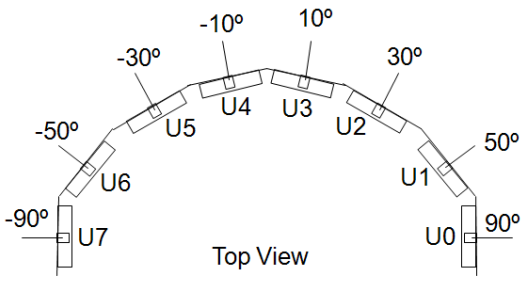


Figure 4.6: Pioneer 2DX sonar array.

The robot has an on-board micro-controller, a $20\ MHz$ Siemens 88C166 microprocessor with integrated 32K Flash-Rom. It also has 32K of RAM, two RS232-compatible serial ports, several digital and analog-to-digital inputs, PSU I/O user-accessible ports and an eight-bit expansible bus. This microprocessor is in charge of controlling all the peripheral electronics, and has its own operating system, P2OS software. It also has an on-board PC104 that connects to the micro-controller through the serial port, which can be replaced by an external PC.

MobileRobots Inc. robots use a client-server robot-control architecture developed by Dr. Kurt Konolige and others at SRI International. The P2OS manages all the low-level details of the robot's system, such as operating the motors, firing the sonar, collecting sonar and wheel encoder data, etc, all on command from and reporting to a separate client application. Every $100\ ms$ it automatically sends an information package via the serial port to the client. This package contains all the sensor readings and parameters relating to the robot's operation. There are a limited number of commands per second that the client can send to the server, but it must be done at least once every two seconds, because if not, the watchdog will stop task execution and the robot's movement.

There are ARIA and Saphira softwares to develop the client program that allow high-level coding but with the power of low level commands. ARIA (ActivMedia Robotics Interface for Application) is an open-source library written in C++ language that allows an easy, high performance access to and management of the robot server, as well as many accessory robot sensors and effectors. Saphira, on the other hand, is a client applications and development environment, developed and maintained by Dr. Konolige. Saphira is per-se a robot control architecture that implements localization, path planning and basic behaviors, as described in Section 2.5.1.

With the tools available to program the mobile robot, it is important to determine its dynamic model in order to develop the right controllers for achieving the proposed tasks. In the next section, the model and the simulators of the robot are explained.

4.5 Robot Model and Simulation System

A dynamic model relates the motion parameters of the robot to applied forces and torques. While forward dynamics is used to analyze the response of the robot to a given command, the inverse dynamic model is used in the design of the controller; so a good dynamic model is a key issue in appropriate control design and simulation of the behavior of the hardware in the control loop.

The dynamics equations of the motion of a mobile robot can be deduced using the Newton-Euler method [102], [105] or the Lagrangian formulation [29, 16, 30, 126, 127, 72, 124]. In this Ph.D. thesis, the Lagrange formalism is used to formulate the dynamics equation of motion of the Pioneer. This equation depends on several physical parameters: some of them are provided by the robot specifications, others can be easily measured and the rest have to be estimated through identification.

Although the complete equation of motion is a complex nonlinear one, when uncoupled experiments (only one degree of freedom is in motion) are carried out, the equation can be simplified and can be formulated as a linear equation in the vector of unknown parameters. Hence, the least square (LS) identification method can be applied to the integral of this equation and the unknown parameters can be estimated. This method, which presents a good statistical behavior, can be used for off-line (simulation) or on-line identification (control) [125]. It has been successfully applied to the parameter identification of an unmanned underwater vehicle [125, 107] and of a Sony SCARA robot [52, 21].

The dynamic model object of study is the Pioneer 2DX mobile robot. The robot moves on a horizontal plane and it is assumed that the wheel-ground contact point satisfies the conditions of pure rolling and non-slipping.

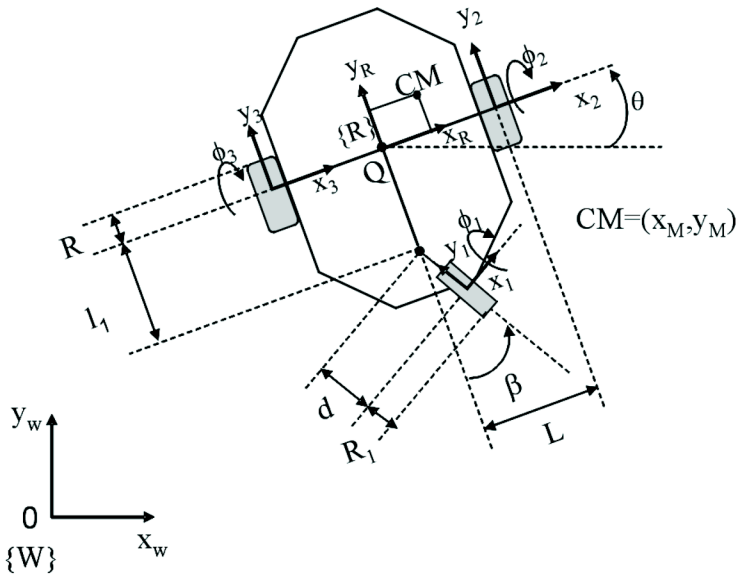


Figure 4.7: Mobile Robot

Figure 4.7 depicts all the robot model's parameters while Table 4.1 shows their significance and their units. The following sections present the equations deduced for the robot model and the motor model.

Table 4.1: Model Parameters.

Symbol	Description	Units
x, y	origin of robot fixed frame	[m]
θ	robot heading	[rad]
β	orientation of the free wheel	[rad]
ϕ_1	angular position wheel 1	[rad]
ϕ_2	angular position wheel 2	[rad]
ϕ_3	angular position wheel 3	[rad]
η_1	linear velocity of robot	[m/s]
η_2	angular velocity of robot	[rad/s]
M	mass of the robot	[kg]
m_2	mass of driving wheel 2	[kg]
m_3	mass of driving wheel 3	[kg]
m_1	mass of free wheel	[kg]
R	radius of wheels 2 and 3	[m]
R_1	radius of wheel 1	[m]
L	distance from robot frame to driving wheels	[m]
l_1	distance from robot frame to the moving bar of the free wheel	[m]
d	distance of the moving bar of the free wheel	[m]
V_a	armature voltage	[V]
i_a	armature current	[A]
x_m, y_m	coordinates of the center of mass (CM)	[m]
I_0	inertia robot	$[kg.m^2]$
I_{r1}	inertia in relation to the rotational axis of the free wheel	$[kg.m^2]$
I_{r2}	inertia in relation to the rotational axis of the driving wheel 2	$[kg.m^2]$
I_{r3}	inertia in relation to the rotational axis of the driving wheel 3	$[kg.m^2]$
I_{p1}	inertia in relation to the vertical axis of the free wheel	$[kg.m^2]$
I_p	inertia in relation to the vertical axis of the wheels 2 and 3	$[kg.m^2]$
C_{s1}	coulomb friction torque free wheel	$[N.m]$
C_{v1}	viscous friction coefficient free wheel	$[kg.m^2/s]$
C_s	coulomb friction torque wheels 2 and 3	$[N.m]$
C_v	viscous friction coefficient wheels 2 and 3	$[kg.m^2/s]$

4.5.1 Robot Model

Let us consider an earth fixed reference frame $\{0, X_w, Y_w\}$ in the plane of motion and a robot fixed reference frame $\{Q, X_r, Y_r\}$ (see Figure 4.7). The vector of generalized coordinates that completely describes the robot motion is:

$$q(t) = (x, y, \theta, \beta, \phi_1, \phi_2, \phi_3) \tag{4.1}$$

Since this system has five independent constraints, it can be proved that it has two degrees of freedom [29]. As proposed by other authors [29, 126, 16], the dynamics equations can be deduced from the Lagrangian formulation:

$$\frac{d}{dt}\left[\frac{\partial[L(q,\dot{q})]}{\partial\dot{q}}\right] - \frac{\partial[L(q,\dot{q})]}{\partial q} = A(q)^T\lambda + B(q)\tau - F_r \tag{4.2}$$

where $L(q, \dot{q})$ is the system's Lagrangian function, equal to kinetic energy minus potential energy. In this particular case, a wheeled mobile robot, the potential energy is zero, so $L(q, \dot{q}) = Ec$ (kinetic energy). As kinetic energy is:

$$Ec = \frac{1}{2}\dot{q}^T M(q)\dot{q} \tag{4.3}$$

Now Equation 4.2 has the form [16], [126]:

$$M(q)\ddot{q} + C(q, \dot{q})\dot{q} = A(q)^T\lambda + B(q)\tau - F_r \tag{4.4}$$

where

- $M(q)$ is a 7x7 positive symmetric inertia matrix,
- $C(q, \dot{q})\dot{q}$ represents the centrifugal and Coriolis forces and torques,
- $A(q)$ is the constraints matrix,
- λ is 5-vector of Lagrangian multipliers associated with the independent kinematic constraints,
- $B(q)$ is the matrix of external torques or forces applied to the robot,
- F_r is the vector of friction forces.

Because of the constraints, there exists a matrix S(q) that satisfies:

$$\dot{q} = S(q)\eta \tag{4.5}$$

where $\eta = \left[\ \eta_1, \eta_2\ \right]^T$.

An important property of S(q) [16], is that

$$S(q)^T A(q)^T = 0 \tag{4.6}$$

This equation allows the elimination of Lagrangian multipliers. Through the time differentiation of Equation 4.5, there can be obtained:

$$\ddot{q} = \frac{d[S(q)\eta]}{dt} = \frac{\partial[S(q)\eta]}{\partial q}\dot{q} + S(q)\dot{\eta} \tag{4.7}$$

By pre-multiplying Equation 4.4 by $S(q)^T$ and replacing $\ddot{q}$ by Equation 4.7, there can be obtained:

$$\begin{gathered} S(q)^T M(q)S(q)\dot{\eta} + S(q)^T M(q)\frac{\partial S(q)}{\partial q}S(q)\eta^2 + S(q)^T C(q, \dot{q})S(q)\eta = \\ S(q)^T A(q)^T\lambda + S(q)^T B(q)\tau - S(q)^T F_r \end{gathered}$$

And the dynamic model written in the space state form is [16]:

$$\left\{ \begin{array}{c} J(q)\dot{\eta} + g(q, \eta) = G(q)\cdot\tau - f_r \\ \dot{q} = S(q)\eta \end{array} \right\} \tag{4.8}$$

where:

$$J(q) = S(q)^T M(q)S(q) \tag{4.9}$$

$$g(q, S(q)\eta) = S(q)^T M(q) \frac{\partial S(q)}{\partial q} S(q)\eta^2 + S(q)^T C(q, \dot{q}) S(q)\eta$$

$$G(q) = S(q)^T B(q) \tag{4.10}$$

$$f_r = S(q)^T F_r \tag{4.11}$$

and:

$$S(q) = \begin{bmatrix} -\sin(\theta) & 0 \\ \cos(\theta) & 0 \\ 0 & 1 \\ -\dfrac{\sin(\beta)}{d} & -\dfrac{d + l_1 \cos(\beta)}{d} \\ \dfrac{\cos(\beta)}{R_1} & -\dfrac{l_1 \sin(\beta)}{R_1} \\ \dfrac{1}{R} & \dfrac{L}{R} \\ \dfrac{1}{R} & -\dfrac{L}{R} \end{bmatrix} \tag{4.12}$$

$$M(q) = \begin{bmatrix} R(\theta)^T M(\beta) R(\theta) & R(\theta)^T V(\beta) & 0 \\ V(\beta)^T R(\theta) & I(\beta) & 0 \\ 0 & 0 & I(\phi) \end{bmatrix} \tag{4.13}$$

$$R(\theta) = \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.14}$$

$$M(\beta) = \begin{bmatrix} M(\beta)_{11} & 0 & M(\beta)_{13} \\ 0 & M(\beta)_{22} & M(\beta)_{23} \\ M(\beta)_{31} & M(\beta)_{32} & M(\beta)_{33} \end{bmatrix} \tag{4.15}$$

$$M(\beta)_{11} = M(\beta)_{22} = M + 2 \cdot m + m_1$$
$$M(\beta)_{13} = M(\beta)_{31} = M \cdot y_m + m_1 \cdot l_1 + m_1 \cdot d \cdot cos(\beta)$$
$$M(\beta)_{23} = M(\beta)_{32} = M \cdot x_m + m_1 \cdot d \cdot sin(\beta)$$
$$M(\beta)_{33} = M(x_m^2 + y_m^2) + 2 \cdot m \cdot L^2 + m_1 \cdot l_1^2 + m_1 \cdot d^2 + 2 \cdot m_1 \cdot l_1 \cdot d \cdot cos(\beta) + I_0 + Ip_1$$

$$V(\beta) = \begin{bmatrix} m_1\, d \cos(\beta) \\ m_1\, d \sin(\beta) \\ Ip_1 + m1\, d^2 + m_1\, l_1\, d \cos(\beta) \end{bmatrix} \tag{4.16}$$

$$I_\phi = \begin{bmatrix} I_{r1} & 0 & 0 \\ 0 & I_{r2} & 0 \\ 0 & 0 & I_{r3} \end{bmatrix} \tag{4.17}$$

$$I_\beta = \begin{bmatrix} m_1\, d^2 + I_{p1} \end{bmatrix} \tag{4.18}$$

$$C(q,\dot{q})\dot{q} = \frac{d[M(q)]}{dt}\dot{q} - \frac{1}{2}\frac{\partial[\dot{q}^T M(q)\dot{q}]}{\partial q} \tag{4.19}$$

$$B = \begin{bmatrix} 0_{5x2} \\ I_{2x2} \end{bmatrix} \tag{4.20}$$

$$F_r = \begin{bmatrix} 0_{3x1} & F_1 & 0 & F_2 & F_3 \end{bmatrix}^T \tag{4.21}$$

$$F_j = C_s sign(\dot{\phi}_j) + C_v\dot{\phi}_j, (j = 2,3) \tag{4.22}$$

$$F_1 = C_{s1} sign(\dot{\beta}) + C_{v1}\dot{\beta} \tag{4.23}$$

F_j and F_1 are the wheels' friction forces or torques [126]. As pure rolling and non-slipping conditions are assumed, these friction torques must be due to the wheels' shaft friction.

4.5.2 Motor Model

In the case of the Pioneer robot, only the voltage applied to the motors can be measured, so the relation between the voltage and the torque must be found.

The torque generated by each motor is:

$$\tau_m = K_T i_a \tag{4.24}$$

while the torque applied to the wheels is:

$$\tau = t_r\mu\{\tau_m - \tau_{f_r}\} \tag{4.25}$$

The applied armature voltage is:

$$V_a = R_a i_a + L_a\frac{di_a}{dt} + K_E t_r\dot{\phi}_i \tag{4.26}$$

$\dot{\phi}_i$ being the speed of the wheel.

Neglecting the inductance L_a:

$$\frac{i_a = V_a - K_E t_r\dot{\phi}_i}{R_a} \tag{4.27}$$

the torque applied to the wheels is:

$$\tau = t_r\mu\{K_T(\frac{V_a - K_E t_r\dot{\phi}_i}{R_a}) - \tau_{f_r}\} \tag{4.28}$$

Parameters are referenced in Table 4.2.

So in Equation 4.8 τ of each motor has to be replaced by Equation 4.28.

Table 4.2: Motor parameters provided by the manufacturer.

Symbol	Parameter	Value	Units
K_T	Torque constant	0.023	$[N.m/A]$
K_E	Back-EMF constant	0.023	$[V/rad/s]$
R_a	Resistance	0.71	$[\Omega]$
L_a	Inductance	Negligible	$[Hy]$
t_r	gear ratio	19.7:1	-
μ	gearbox efficiency	73	%
τ_{f_r}	friction torque	$5.6 * 10^{-3}$	$[N.m]$

4.5.3 Identification method

If the equation of a model can be expressed as:

$$\dot{\eta} = \Psi(\eta, \tau)\Theta \tag{4.29}$$

then the parameters vector can be estimated using the least square method, as the system is linear in respect of the parameters vector [125], $\Psi(\eta, \tau)$ being a matrix whose values only depend on the state and control vectors and Θ the vector of unknown parameters. The objective of the least square method is to minimize the cost function of the quadratic error. It can be proven [125] that by integrating both sides of Equation 4.29 between t_{k-1} and t_k:

$$\eta(t_k) - \eta(t_{k-1}) = \int_{t_{k-1}}^{t_k} [\Psi(\eta(t), \tau(t))dt]\Theta \tag{4.30}$$

the values of the parameters that minimize the cost function are obtained as:

$$\Theta(N) = (F(N)^T F(N))^{-1} F(N)^T Y(N) \tag{4.31}$$

where:

$$F(N) = \begin{bmatrix} \int_{t_0}^{t_1} [\Psi(\eta(t), \tau(t))dt] \\ \int_{t_1}^{t_2} [\Psi(\eta(t), \tau(t))dt] \\ \dots \\ \int_{t_{N-1}}^{t_N} [\Psi(\eta(t), \tau(t))dt] \end{bmatrix}$$

$$Y(N) = \begin{bmatrix} (\eta(t_1) - \eta(t_0)) & \dots & (\eta(t_N) - \eta(t_{N-1})) \end{bmatrix}^T$$

Using this algorithm the noise introduced by the calculation of the acceleration [21] is avoided.

Mobile Robot Application

To identify the parameters, Equation 4.8 can be rewritten as:

$$\dot{\eta} = J(q)^{-1}[G(q) \cdot \tau - g(q, S(q)\eta) - f_r] \tag{4.32}$$

Or in a compact form:

$$\begin{aligned}\dot{\eta}_1 &= a_{11}\cdot F + a_{12}\cdot\Gamma - a_{13}\cdot\eta_1^2 - a_{14}\cdot\eta_2^2 - a_{15}\cdot\eta_1 - a_{16}\cdot\eta_2 - a_{17}\cdot\eta_1\cdot\eta_2 - a_{18}\cdot sign(\eta_1) - a_{19}\cdot sign(\eta_2)\\ \dot{\eta}_2 &= a_{21}\cdot F + a_{22}\cdot\Gamma - a_{23}\cdot\eta_1^2 - a_{24}\cdot\eta_2^2 - a_{25}\cdot\eta_1 - a_{26}\cdot\eta_2 - a_{27}\cdot\eta_1\cdot\eta_2 - a_{28}\cdot sign(\eta_1) - a_{29}\cdot sign(\eta_2)\end{aligned} \tag{4.33}$$

where coefficients a_{ij} are a non-linear combination of the physical parameters of the robot.

In order to simplify the identification process, experiments can be carried out for each degree of freedom. For the linear movement ($\eta_2 = 0$ and $\beta = 0$), Equation 4.33 becomes:

$$\dot{\eta}_1 = a_{11}\cdot F - a_{15}\cdot\eta_1 - a_{18}\cdot sign(\eta_1) \tag{4.34}$$

and:

$$F = \frac{t_r\cdot\mu\cdot K_T\cdot(V_2+V_3)}{R\cdot R_a}$$

$$a_{11} = \frac{1}{(M+2\cdot m+m_1+\frac{I_{r1}}{R_1^2}+\frac{2\cdot I_r}{R^2})}$$

$$a_{15} = \frac{2\cdot(t_r^2\cdot\mu\cdot K_T\cdot K_E+R_a\cdot C_v)}{R^2\cdot Ra}\cdot a_{11}$$

$$a_{18} = \frac{2\cdot[C_s+t_r\cdot\mu\cdot\tau_r]}{R}\cdot a_{11}$$

For the angular movement ($\eta_1 = 0$ and $\beta = -90^o$), Equation 4.33 becomes:

$$\dot{\eta}_2 = a_{22}\cdot\Gamma - a_{26}\cdot\eta_2 - a_{29}\cdot sign(\eta_2) \tag{4.35}$$

where:

$$\Gamma = \frac{L\cdot t_r\cdot\mu\cdot K_T\cdot(V_2-V_3)}{R\cdot R_a}$$

$$a_{22} = \frac{1}{[M\cdot(x_m^2+y_m^2)+I_0+2\cdot I_p+2\cdot m\cdot L^2+m_1\cdot l_1^2+\frac{l_1^2\cdot I_{r1}}{R_1^2}+\frac{2\cdot L^2\cdot I_r}{R^2}]}$$

$$a_{26} = \frac{(2\cdot L^2\cdot t_r^2\cdot\mu\cdot K_T\cdot K_E+C_{v1}\cdot R^2\cdot R_a+2\cdot L^2\cdot C_v\cdot R_a)}{R^2\cdot R_a}\cdot a_{22}$$

$$a_{29} = \frac{(2\cdot L\cdot t_r\cdot\mu\cdot\tau_r+C_{s1}\cdot R+2\cdot L\cdot C_s)}{R}\cdot a_{22}$$

As equations 4.34 and 4.35 are linear with respect to a_{ij}, then the method explained at the beginning of this section can be applied.

The unknown parameters to be identified are: I_0, I_{r1}, $I_r = I_{r2} = I_{r3}$, I_{p1}, C_{s1}, C_{v1}, C_s and C_v.

PARAMETER IDENTIFICATION

The method has been applied to the Pioneer robot. Some known characteristics are shown in Table 4.3.

Table 4.3: Known Characteristics.

Parameter	Value	Units
R	0.0825	m
R_1	0.035	m
L	16.89	m
l_1	0.18	m
d	0.03	m
x_m	0	m
y_m	-0.07	m
M	15.5	[kg]
m	0.35	[kg]
m_1	0.35	[kg]

Measured variables: Voltage applied to the V_2, V_3 motors and linear η_1 and angular η_2 speeds. Velocities are provided directly from the software of the robot, while for voltages an external board is used to adapt them to the analog inputs of the micro-controller.

The inputs used are STEP and PRBS signals. The data that is gathered is validated in order to eliminate outliers and trends and filtered to reduce the effects of noise.

For each degree of freedom, three experiments at different speeds were carried out. In the case of linear movement, results found are shown in Table 4.4. The last row represents the mean value for each parameter estimated.

Table 4.4: Translational movement.

Exp.	$1/a_{11}$	C_v	C_s	Input
1	17.83	0.0441	0.3349	PRBS
2	16.49	0.0445	0.3287	PRBS
3	16.89	0.0446	0.2921	STEP
Mean	17.07	0.0444	0.3186	

In order to validate the results, another set of experiments was carried out. Figure 4.8 depicts the input signal used for one experiment, a step forward and backward. Figure 4.8 a) shows the input voltage applied to each motor, Figure 4.8 b) the total force applied to the robot and Figure 4.8 c) the linear speed of the vehicle. The performance of the model is presented in Figure 4.9, while the statistical validation is shown in Figure 4.10. Figure 4.10 a) shows the residuals, Figure 4.10 b) the histogram of the residuals, which have a mean value of $-1.4864e^{-4}$, a standard deviation of 0.0367 and are approximately Gaussian. Figure 4.10 c) shows the autocorrelation of the residuals.

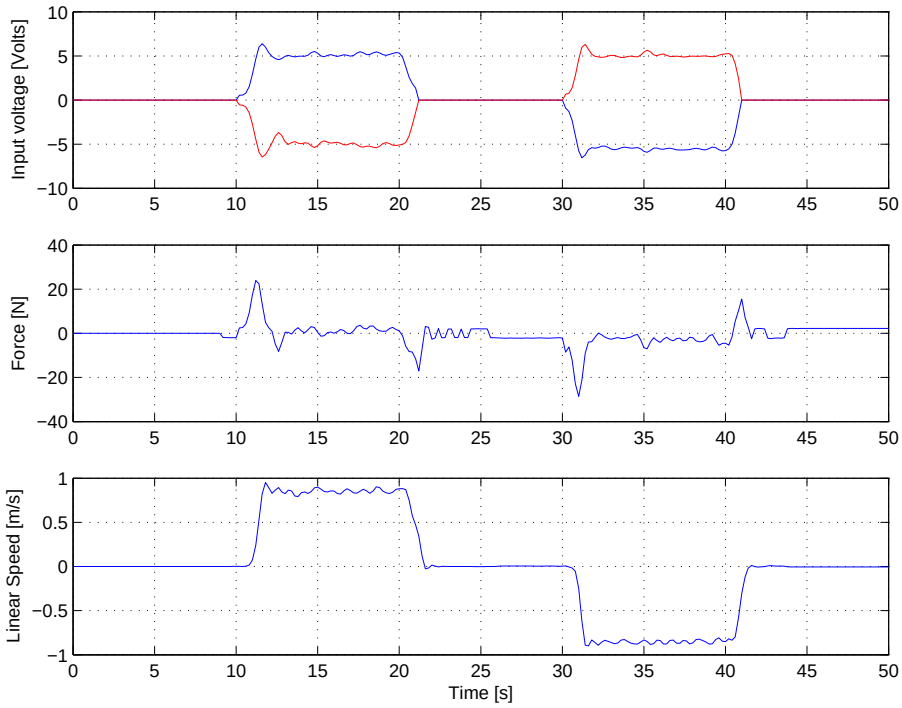


Figure 4.8: Input and output signals from the robot.

By applying the same ideas to rotational movement, Table 4.5 is obtained.

Table 4.5: Rotational movement.

Exp.	$1/a_{22}$	C_{v1}	C_{s1}	Input
1	0.4301	0.000429	0.004535	STEP
2	0.4127	0.000389	0.004532	PRBS
3	0.4292	0.000469	0.004541	PRBS
Mean	0.424	0.000429	0.004536	

With this method the values for the combination of the model parameters are found. As the purpose of this work is to find a model suitable for simulation, the exact value of these parameters has been deduced from the theoretical value. For example:

$$\frac{1}{a_{11}} = M + 2 \cdot m + m_1 + \frac{I_{r1}}{R_1^2} + \frac{2 \cdot I_r}{R^2} = 15.5 + 2 \cdot 0.35 + 0.35 + \frac{2.144 \cdot 10^{-4}}{0.035^2} + \frac{2 \cdot 0.0012}{0.0825^2} = 17.0750$$

which is quite similar to the value found empirically.

The estimated parameters for both degrees of freedom are shown in Table 4.6. It is worth noting that even though an uncoupled identification of the parameters has been carried out, all the coefficients of the coupled model were found.

As can be seen from Figure 4.10, the residuals are Gaussian and the mean value is approximately zero. These results can be considered statistically acceptable. The parameter value shown in Table 4.6 is the mean of all the different values obtained in each experiment.

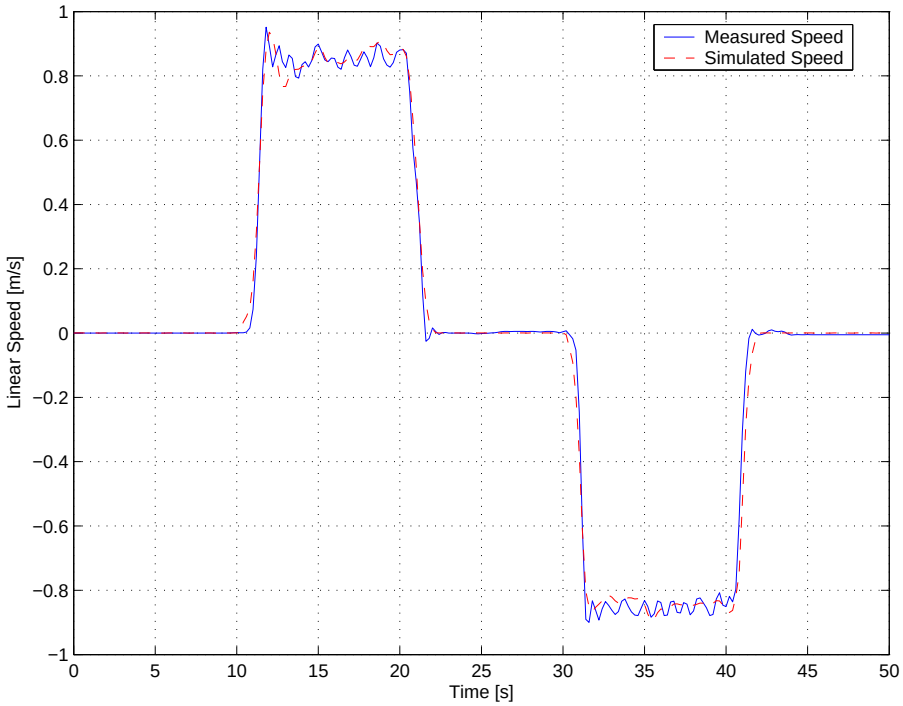


Figure 4.9: Response speed of the robot and the model.

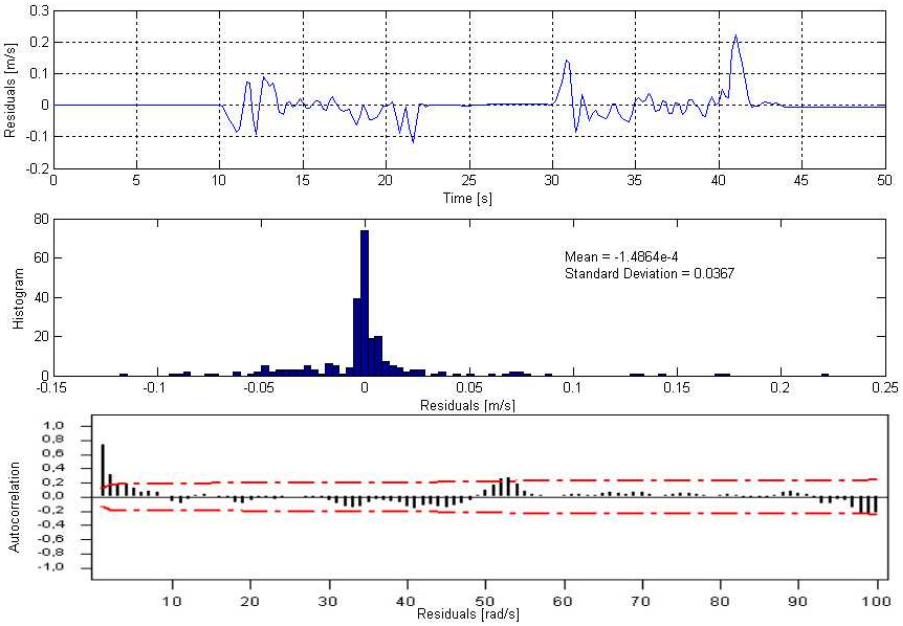


Figure 4.10: Residuals, their histogram and their autocorrelation.

Table 4.6: Estimated parameters.

Symbol	Value	Units
I_0	0.3	$[kg.m^2]$
I_{r1}	$2.144 * 10^{-4}$	$[kg.m^2]$
I_{r2-3}	0.0012	$[kg.m^2]$
I_{p1}	$1.072 * 10^{-4}$	$[kg.m^2]$
I_p	$5.95 * 10^{-4}$	$[kg.m^2]$
C_{s1}	0.004536	$[N.m]$
C_{v1}	0.000429	$[N.m.s]$
C_s	0.3184	$[N.m]$
C_v	0.0444	$[N.m.s]$

4.5.4 Simulation System

After calculating the robot model and parameter identification, a simulator was built using Simulink - Matlab$^{(R)}$. This simulator has a 3D interface to visualize the robot's movements. Several views of the simulator are shown in Figure 4.11. The block diagram of the dynamic model of the robot is depicted in Figure 4.12.

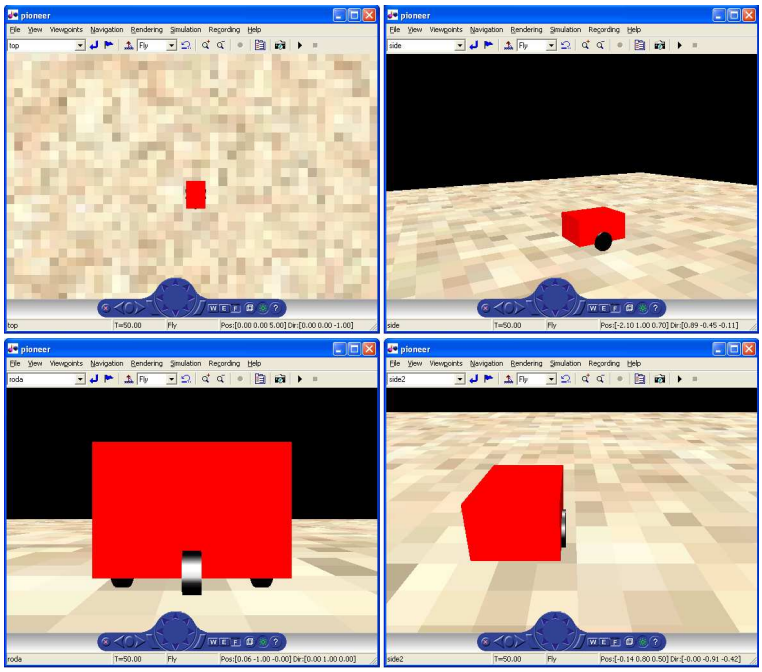


Figure 4.11: 3D Simulator in Matlab$^{(R)}$.

This model is the implementation of the equations described in the previous subsections. The complete robot model, including the speed controllers running in the micro-processor, is shown in Figure 4.13. This model was used to calculate and simulate the different controllers of the behavioral agents.

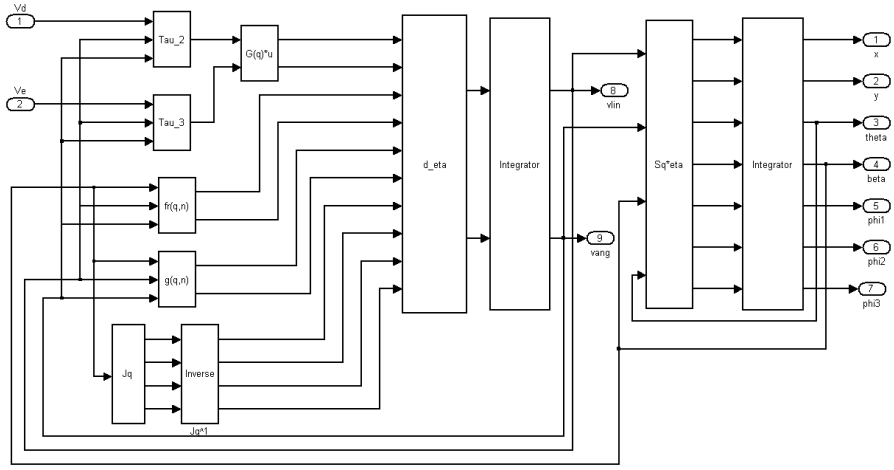


Figure 4.12: Robot dynamic model in Simulink$^{(R)}$.

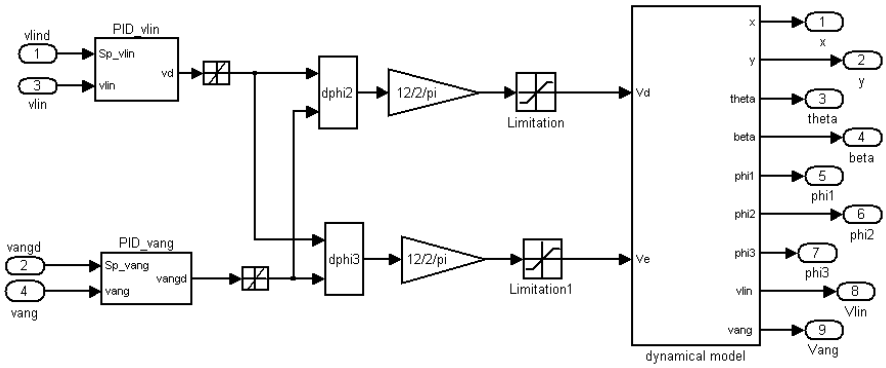


Figure 4.13: Robot dynamic model in Simulink$^{(R)}$ with the speed controllers.

One of the disadvantages of the 3D simulator is in building complex environments, so when a new and more accurate simulator was provided by MobileRobots Inc. the 3D simulator was discarded and MobileSim adopted to test ARMADiCo.

MobileSim can simulate MobileRobots Inc. mobile robots and their environments. It can be used to experiment and debug client programs that utilize ARIA. It is based on the Stage library, created by the Player/Stage/Gazebo project [68] (details of ARIA and the Player/Stage project are in subsection 2.7.6). MobileSim is a 2D simulator that converts MobileRobots map files into Stage environment and places a simulated robot model in this environment. This simulator allows the customizing of robot models and device models such as sonar, laser, etc. The most important advantage of this simulation software is that once the client program has been tested and debugged, it can be used to control the robot directly, since it is possible to automatically program the connection to the robot if present or to the simulator in the code of the client application.

Figure 4.14 shows the main window of the simulator. It connects to the client program via a TCP connection and with the real robot using the serial port.

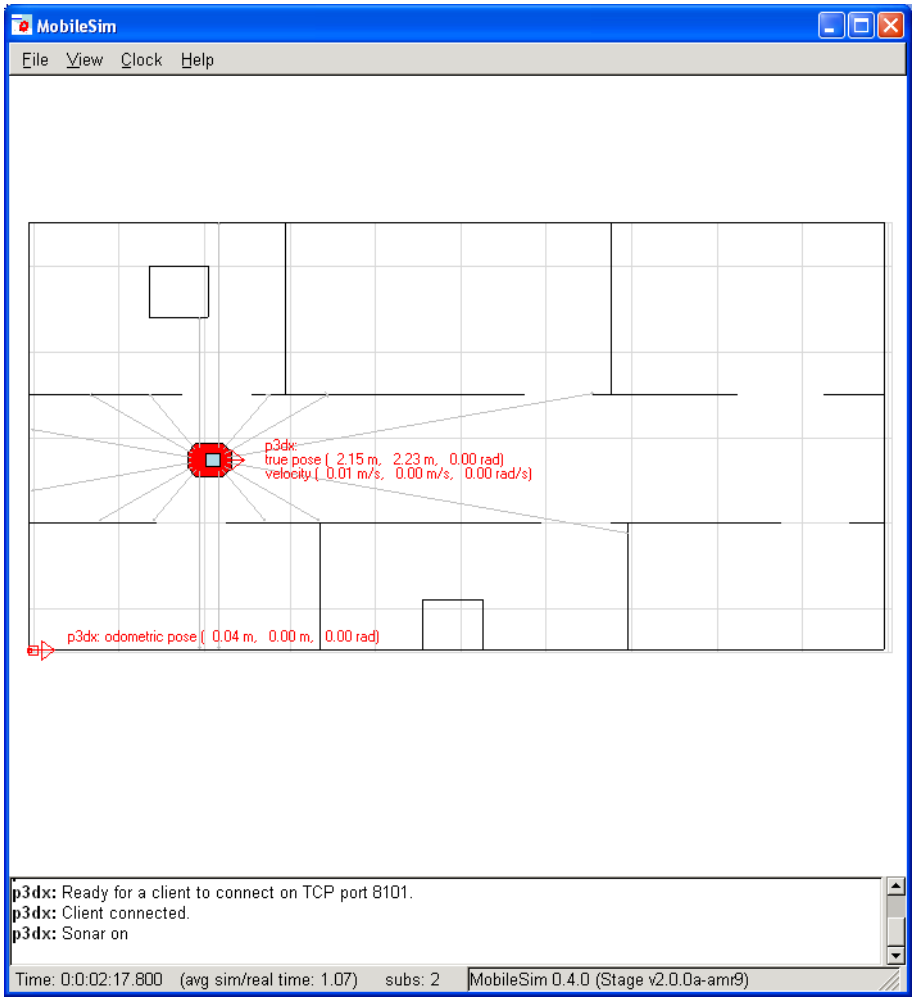


Figure 4.14: MobileRobots Inc. MobileSim simulator.

4.6 Collaborative Controllers

This section presents the controllers employed to strengthen the cooperative method based on the fuzzy logic of the *goto* agent. Instead of developing only one very elaborate controller, several simple controllers have been designed to deal with different aspects of the control separately and unify their actions to obtain a final complex behavior. The Sugeno fuzzy inference methodology provides suitable way of modeling the importance of the various controllers as explained in Section 3.5.1.

The block diagram of cooperative control proposed for the *goto* agent is shown in Figure 3.12. The fuzzy collaborative controller block is in charge of the mixing of the desired controller velocities to convert progressively from one controller to another.

4.6.1 Speed Controllers

In order to obtain the desired angular and linear speeds, two independent PI controllers were developed, one for angular speed and the other for linear speed. One particular aspect of these controllers is that, instead of using T-approximation (truncation of the control

vector), they use the S-approximation (scaling) to obtain feasible control vectors **u(t)** that do not violate constraints (see [96] for details). As the resulting feasible vector is proportional to the unfeasible one, both have the same direction and there is no error in the desired orientation.

Figure 4.15 shows the results for several set-points of angular and linear speeds. Figure 4.15-a) shows the response of the robot moving forwards, Figure 4.15-b) rotation through its vertical axis, and Figure 4.15-c) a coupled movement. Analysis of the response of the controllers for different set-points, shows that a steady state is reached in under two seconds and the overshoot is less than 10% in the worst case.

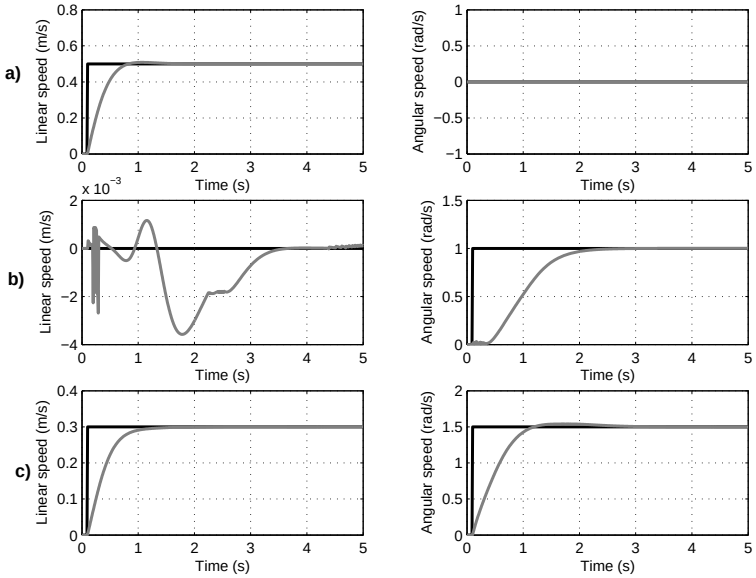


Figure 4.15: Response of the speed controllers for different set-points.

4.6.2 Fast Position Controller

The idea is to obtain a positional controller that is as fast as possible, no matter whether the set-point is reached or not. Figure 4.16 shows the response of the controlled system for the initial position and heading of $(x_0, y_0, \theta_0) = (0, 0, 0)$ and the desired set-point of $(x_f, y_f, \theta_f) = (-1, 5, 0)$. Figure 4.16-a) and Figure 4.16-b) show the evolution of the x and y coordinates over time, Figure 4.16-c) the robot heading over time, Figure 4.16-e) and Figure 4.16-f) the linear and angular speeds and Figure 4.16-d) the robot trajectory ($y = x(t)$). As shown in Figure 4.16-d), the robot does not reach the desired position exactly, and even the heading is not the desired point, but it is fast: it stops in five seconds.

4.6.3 Slow Position Controller

Here, the goal is to obtain a position controller that arrives at the desired position and heading regardless of the time taken to reach the set-point. Figure 4.17 presents the response of the controlled system for the same initial and final position and heading of the previous experiment, that is,$(x_0, y_0, \theta_0) = (0, 0, 0)$ and $(x_f, y_f, \theta_f) = (-1, 5, 0)$. Figure 4.17-a) and

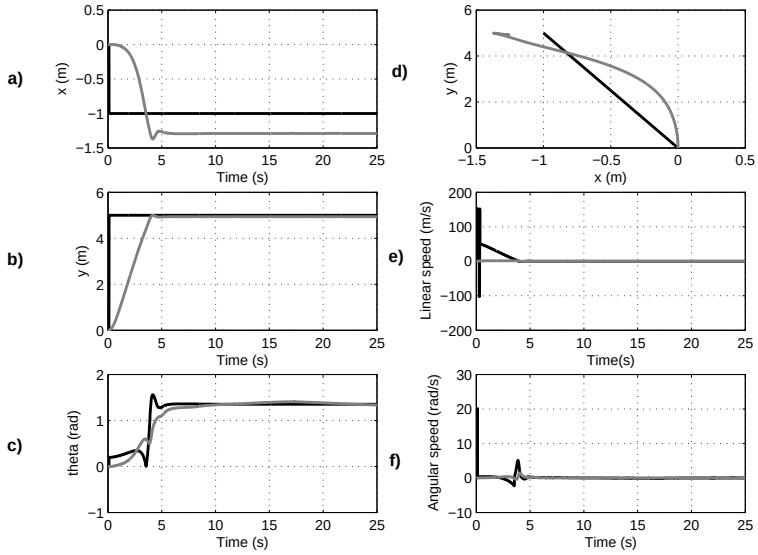


Figure 4.16: Response of the fast PID controller.

Figure 4.17-b) show the evolution of the x and y coordinates over time, Figure 4.17-c) the robot heading, Figure 4.17-e) and Figure 4.17-f) the linear and angular speeds and Figure 4.17-d) the robot trajectory ($y = x(t)$). As can be seen, the robot takes more than 10 seconds to arrive at the set-point. This is twice the time needed by the fast controller. However, the fast controller leads to an erratic final position, while the accurate control achieves the goal coordinates exactly.

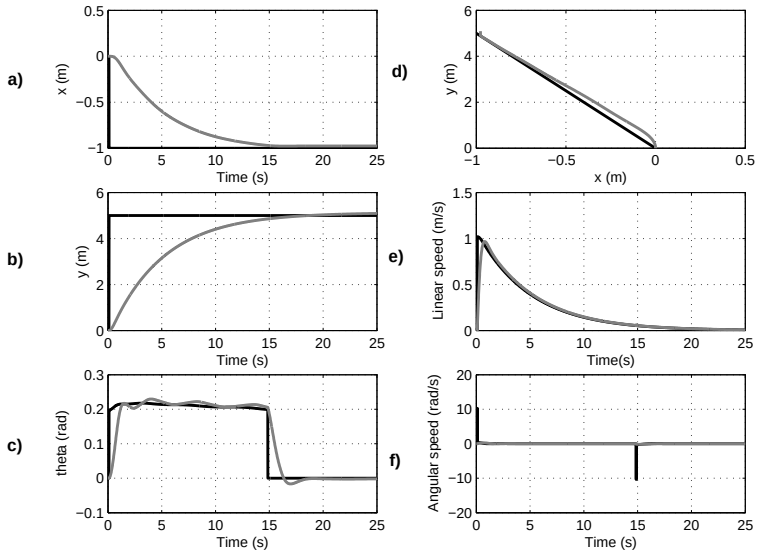


Figure 4.17: Response of the slow PID controller.

4.6.4 Fuzzy Adjustment Collaborative Control

Figure 4.18 shows the response of the whole system using the proposed collaborative control (see Equation (3.1)) for the same initial and final position and heading as in the previous experiments. As can be seen, the robot reaches the desired set-point faster than with the slower controller and more accurately than with the faster controller.

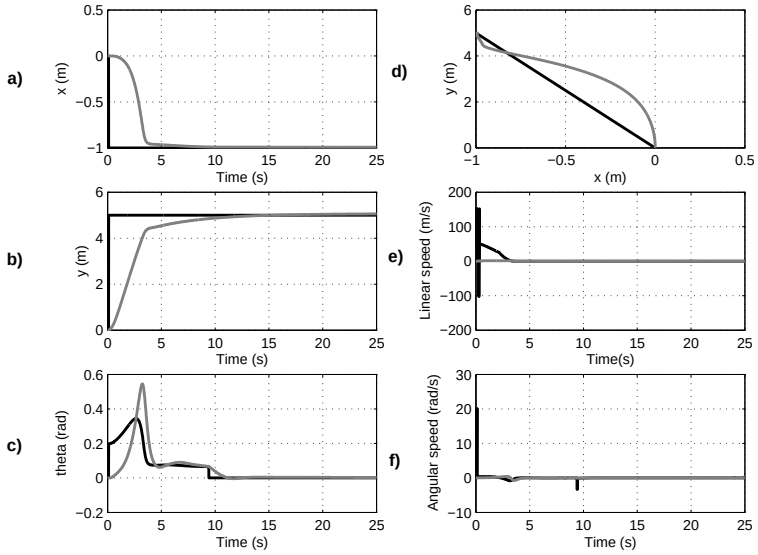


Figure 4.18: Response of the fuzzy concurrent control.

One interesting feature of this collaborative controller is that it works better than the controllers working separately, especially for the unreachable states produced by the non-linear nature of the robot model. For example, considering the initial state as $(x_0, y_0, \theta_0) = (0, 0, \frac{-3 \cdot \pi}{8})$ and a final state as $(x_f, y_f, \theta_f) = (-2, 3, \frac{-\pi}{2})$, this is only achieved by the concurrent control, as depicted in Figure 4.19. The results shown in Figure 4.19-b) for the faster controller are expected because it was designed to be fast but not accurate. Figure 4.19-a) shows the response of the slower controller; although it was intended to be accurate, it cannot arrive at the desired set-point due to non-linearities. This could be solved by improving the controller design (e.g. non-linear controller) or using concurrent control, as depicted in Figure 4.19-c). The results obtained show that collaborative control is precise, while maintaining responsiveness in the overall multi-agent architecture.

4.7 Conclusions

In this chapter all the components necessary to build an ARMADiCo platform and agents are given. First, the multi-agent engineering methodology on which the design of ARMADiCo is based are discussed and some results of the different phases are presented. Using this methodology helps to obtain a coherent multi-agent model where errors in the design are eliminated and some important features, such as the coordination protocol, can be defined before implementation.

Next, the multi-agent platform used to implement ARMADiCo is described, highlighting the most relevant features of the ad-hoc platform. The decision to build an ad-hoc platform was made based on the facts that at the time when the system was conceived, commercial multi-agent platforms did not support real-time constraints and there were agents that centralized

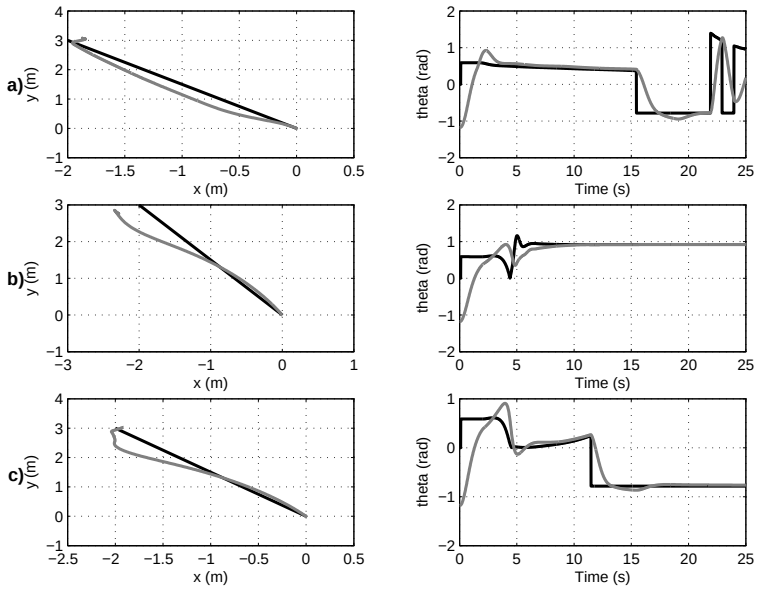


Figure 4.19: Comparison of the response of the controllers; a) the slow controller, b) the fast controller and c) the concurrent control.

the whole system. This ad-hoc platform allows new agents to be easily created and the agents in the platform follow the structure of the agent design pattern given in the previous chapter. Some of the pattern's methods are already implemented in the basic agent but it is possible to modify or change them in the new instance of the agent. Others must be programmed in each instance (as in the goal method). Also, steps such as registering or creating communication channels are automatic, depending on the declaration of provided services, requested services and resource sharing. In this platform, agents are individually identified using an agent name, a host name and a port number. With this feature, it is possible to distribute agents in different computers. Finally, it is also possible to use different programming languages for the agents, provided that the communication protocol and message content are in accordance with the specifications.

In order to build the different agents of ARMADiCo, it is necessary to know the hardware platform: available sensors, actuators and the interaction between them. A description of the Pioneer 2DX is therefore presented, highlighting the operating system of the on-board micro-processor, which is responsible for acquiring sensor information and controlling all the associated hardware, including sensors. This operating system acts as a server and in order to control the robot, a client application must be created. This client application can be programmed using ARIA, a high level library that allows low level performance to be obtained without having to concern itself with low level commands.

With the identity of the mobile robot established, it now is important to learn about the way it behaves, i.e. the laws that govern its movements of the robot. This is known as the dynamic model. A brief description of the mathematical equations of these movements has been given, as well as the identification method used to obtain all the parameters involved in the equations. With this mathematical model, the controllers of the different behavioral agents were been tuned and a simulator was been developed to test them. This simulator was developed in Matlab-Simulink$^{(R)}$ and visualization of the movement was performed in 3D. Unfortunately, it had to be discarded, because complex environments were difficult to simulate and the coding of agents was not suitable for the real platform. For these reasons, and benefiting from the new and more accurate software simulator provided by

MobileRobots Inc., the MobileSim simulator was adopted to test the multi-agent platform.

Finally, the collaborative controllers of the goto agent which are mixed with fuzzy logic, are presented. The resulting control system has more benefits over the single controllers, since the final result is more precise and faster than the one obtained with the separate controllers. Another important factor is that there are some states (due to the non-linear nature of the robot) that are only reachable using the collaborative controller.

In the next chapter, ARMADiCo is tested in several scenarios using the configurations described in this chapter in order to analyze its emergent behavior in different situations.

CHAPTER 5

Experimental Results

"They didn't understand what they were doing". I'm afraid that will be on the tombstone of the human race. I hope it's not. We might get lucky.

From *Prey* by Michael Crichton

In this chapter, a number of results are presented. First, the collaborative control is tested as it interacts with the other agents in the architecture. Next, the winner determination algorithm (explained in Section 3.10.1) is featured. Then, the resource exchange method is tested in two different situations, with abrupt change and using the proposed fuzzy method (described in Section 3.10.2). Next some results adding a new agent are presented and finally the interactions between the individual and social intelligences of an agent are shown.

5.1 INTRODUCTION

An architecture can be evaluated according to certain criteria an in [89] there is specified a set of properties with which to do so. They are:

- **Support for modularity:** refers to good software engineering principles.
- **Niche targetability:** to determine how well the architecture works for the intended application.
- **Ease of portability to other domains:** this means the quality of the architecture when working in other applications or other robots.
- **Robustness:** refers to vulnerability in the system and how it is reduced.

These properties indicate how good and/or versatile an architecture can be. Often these properties are at odds with each other, as for example the niche targetability and the ease of portability. Most of the architectures studied in Chapter 2 choose the portability property.

Therefore, to test the support for modularity, it is proposed to first add or remove agents from ARMADiCo. To this end, the global behavior of ARMADiCo is compared in two different configurations, the first one having only the goto and avoid agents, and the second with the gothrough agent added. Second, with regard to niche targetability several experiments in different scenarios are proposed: first, collaborative control is tested when the goto agent

is interacting with the other agents in the architecture (at the beginning only with the avoid agent, but then with the gothrough agent, as well). With regard to distributed coordination, the utility function is analyzed in two different scenarios to show the emergent behavior of the control architecture; the agent design pattern is tested by adding the gothrough agent to the control architecture and analyzing the changes that have to be performed on the other agents; and finally emergent behavior inside the agent (interactions between individual and social intelligences) and the resource exchange method are tested. Finally, portability and robustness is left for future work.

In addition to these experiments, it would be useful to compare ARMADiCo against other robot architectures, so that the progress that has been made in robotics could be properly confirmed but unfortunately such a comparison is very difficult. It would mean the re-implementation of some state-of-the-art architectures, a difficult undertaking. Usually, and most importantly, the experiments carried out to test the architectures are unrepeatable because scenarios are hard to duplicate, robots are not the same, and in cases where they are, their controllers can be difficult to tune with the same parameters as the original, meaning that the original experiment can not be reproduced. Also, the programming code of the architectures is usually not available, or the parameters of the architecture itself are tuned for a specific robot application.

In order to compare the different experiments carried out with ARMADiCo, the following measurements have been defined:

- **Traveled Distance (TD):** the distance traveled by the robot to reach the goal.
- **Final Orientation (FO):** the heading of the robot at the goal position.
- **Total Time (TT):** the total amount of time the robot needs to achieve its goal.
- **Precision (P):** how close to the goal position is the the center of mass of the robot.
- **Time Goto (TG):** the total time the *goto* agent has control of the robot.
- **Time Avoid (TA):** the total time the *avoid* agent has control of the robot.
- **Time Gothrough (TGt):** the total time the *gothrough* agent has control of the robot.

These measurements give quantitative values with which to compare the different experiments and to analyze the improvements of the whole architecture.

To perform the experiments a minimum configuration of ARMADiCo has been implemented; the running agents are:

- **Social agents:** interface agent.
- **Deliberative agents:** A very simple path planning agent.
- **Behavioral agents:** goto, avoid and gothrough agents.
- **Perception agents:** encoder, sonar and battery Sensor agents.
- **Actuator agents:** robot agent.
- **Back agents:** DF, wakeup, monitor agents.

This minimal configuration varies only with the addition or deletion of certain agents according to the purpose of the experiment.

5.2 Simulated Results

5.2.1 Collaborative Control

The hypothesis is that the use of a collaborative controller in the multi-agent architecture improves robust behavior in respect of other controllers, even when there are interactions with other agents. Four scenarios are used to test the performance of the control architecture. They are shown in Figures 5.1, 5.2, 5.3 and 5.4. To carry out the experiments, three different versions were implemented for the goto agent: the first uses only the fast controller, the second uses the precise controller and the third version uses fuzzy cooperative control.

Scenario 1 (Sc1): In this scenario, the robot must go from the initial position $(x_0, y_0, \theta_0) = (0.0, 0.0, 0.0)$ to a target position $(x_f, y_f, \theta_f) = (1.5, 1.0, 0.0)$, the position being expressed in meters and the orientation in degrees, as can be seen in Figure 5.1. In this scenario there are no obstacles present in the path of the robot, so the avoid agent does not intervene.

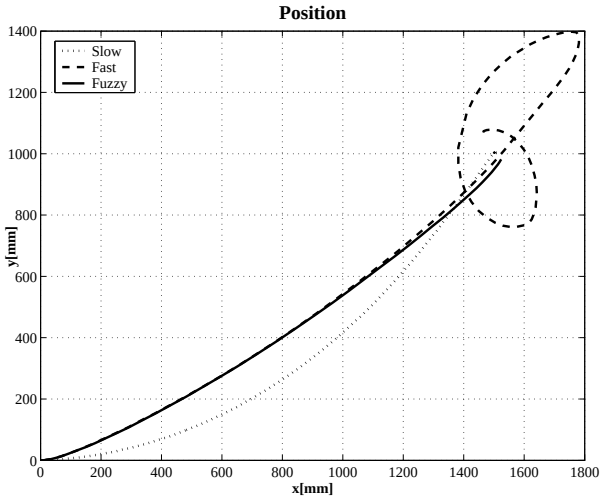


Figure 5.1: Example of the trajectory for the three goto agent implementations in Scenario 1: trajectory without obstacles

Scenario 2 (Sc2): In this setting, an obstacle is introduced into the path of the robot and so, the avoid agent intervenes in its guidance. The initial position is $(x_0, y_0, \theta_0) = (0.0, 0.0, 0.0)$ and the final is $(x_f, y_f, \theta_f) = (10.0, 0.0, 0.0)$ as can be seen in Figure 5.2. There is a wall $2.56m$ long located $5m$ from the robot.

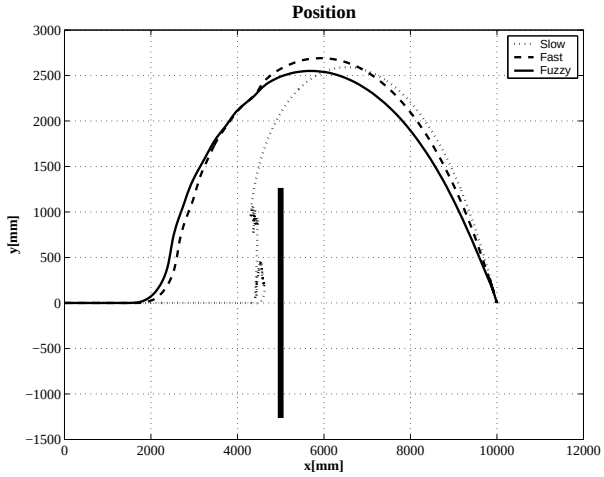


Figure 5.2: Example of the trajectory for the three goto agent implementations in Scenario 2: trajectory with simple obstacle

Scenario 3 (Sc3): In this example, the robot moves through a corridor with a column in the middle. The initial position of the robot is $(x_0, y_0, \theta_0) = (1.0, 0.5, 0.0)$, the final position is at $(x_f, y_f, \theta_f) = (5.0, 0.6, 0.0)$, as can be seen in Figure 5.3. Again, the avoid agent intervenes in the robot's guidance.

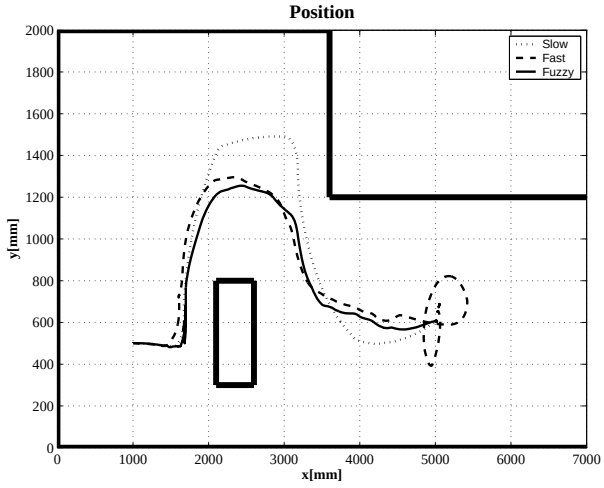


Figure 5.3: Example of the trajectory for the three goto agent implementations in Scenario 3: moving through a corridor

Scenario 4 (Sc4): In this scenario, the robot must go from room A to room B. In this case, the path planning agent intervenes in the calculation of the trajectory, giving the goto agent the points the robot needs in order to pass through the doors, as can be seen in Figure 5.4.

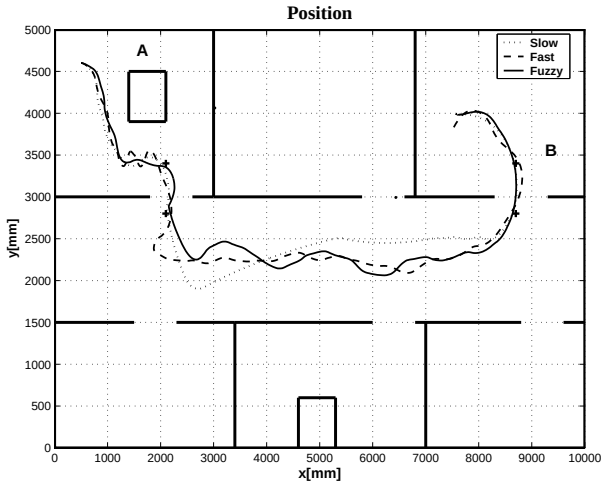


Figure 5.4: Example of the trajectory for the three goto agent implementations in Scenario 4: across rooms

The experiments were carried out separately for each implementation of the goto agent and all the results are compared according to the measures previously defined.

The experiments consisted of five executions in each scenario for each implementation of the goto agent. Table 5.1 shows the average and standard deviation of each evaluation measure and scenario. With regard to the statistical significance of the results obtained, the p values (with $t_{0.05}$) are the ones expected.

The traveled distance (**TD**) has the lowest value for the fuzzy collaborative controller except in Scenario 4. To explain this behavior, one needs to look at the particular trajectories described by the robot. In Figure 5.1 it can be observed that the cooperative control behavior follows the fast controller at the beginning and the precise one at the end, thereby obtaining a curve which comes close to the ideal (which would be a straight line between the initial and the final points). This happens in the other scenarios too, but the interaction with the avoid agent hides it. In Scenario 4, the path planning agent gives several points between the initial position and the desired goal. These points, marked with an x in Figure 5.4, are situated near the doors. In this case, the collaborative controller follows both controllers several times during the execution of the trajectory.

The total time (**TT**) consumed by the precise controller to reach the goal position is the greatest. The collaborative controller times are similar to those of the fast controller, meaning that its benefits are being obtained.

Even though the precision (**P**) of all the controllers is above 90%, the traveled distance (**TD**) approximates more to the minimum distance (i.e. a straight line to the goal position) in all the scenarios with the collaborative controller. On the other hand, the final orientation (**FO**) of the collaborative controller (as well as of the precise controller) approximates better to the desired orientation.

With regard to the time goto (**TG**), its value is higher with the precise controller because, as it is slow, the avoid agent does not need to take control of the robot very often only when the obstacle is very near the robot. This is not true for Scenario 2 where the obstacle is so long that the avoid agent needs more time to dodge it. With the fast controller it is just the contrary: the avoid agent needs to intervene more often. The exception is again Scenario

Table 5.1: Comparison of the *goto agents* for the three different controllers.

Controller	Parameters	Sc1	Sc2	Sc3	Sc4
Fast	TD [m]	5.55 ± 0.43	13.37 ± 0.40	6.10 ± 1.38	15.71 ± 2.85
	FO [°]	7.58 ± 4.71	49.18 ± 44.56	25.96 ± 29.76	56.75 ± 67.98
	TT [s]	14.28 ± 1.29	27.78 ± 3.80	28.20 ± 6.67	64.66 ± 13.5
	P [%]	99.46 ± 0.56	99.36 ± 0.25	97.83 ± 0.49	94.40 ± 5.78
	TG [%]	100 ± 0	90.19 ± 2.82	50.07 ± 5.23	50.22 ± 4.89
	TA [%]	0 ± 0	9.81 ± 2.82	49.93 ± 5.23	49.78 ± 4.89
	TGt [%]	-	-	-	-
Precise	TD [m]	5.17 ± 0.05	17.92 ± 2.84	5.78 ± 0.38	12.26 ± 0.10
	FO [°]	1.37 ± 1.13	1.77 ± 2.93	2.58 ± 0.90	2.97 ± 2.93
	TT [s]	34.50 ± 6.25	164.40 ± 20.91	40.62 ± 4.48	79.00 ± 2.77
	P [%]	99.81 ± 0.16	99.85 ± 0.07	99.83 ± 0.10	99.79 ± 0.11
	TG [%]	100 ± 0	80.82 ± 5.06	81.95 ± 3.20	85.99 ± 1.53
	TA [%]	0 ± 0	19.18 ± 5.06	18.05 ± 3.20	14.01 ± 1.53
	TGt [%]	-	-	-	-
Collaborative	TD [m]	5.11 ± 0.01	12.46 ± 0.66	5.08 ± 0.30	12.55 ± 0.44
	FO [°]	1.95 ± 0.71	1.76 ± 0.84	1.97 ± 1.01	1.77 ± 1.63
	TT [s]	16.56 ± 0.44	27.76 ± 1.22	27.90 ± 1.35	66.06 ± 8.79
	P [%]	99.41 ± 0.22	99.67 ± 0.06	99.55 ± 0.16	99.78 ± 0.13
	TG [%]	100 ± 0	88.40 ± 2.29	58.68 ± 4.27	66.79 ± 2.31
	TA [%]	0 ± 0	11.6 ± 2.29	41.32 ± 4.27	33.21 ± 2.31
	TGt [%]	-	-	-	-

2; as this controller is very fast, both agents interact to dodge the obstacle faster, so the goto agent has control more often. For the collaborative controller this measurement is between the two values, the **TG** of the fast controller and the **TG** of the precise controller, as expected.

With regard to the time avoid (**TA**), should to be noted that in this particular case, as there is no other behavioral agent in the current experiment, it complements the **TG**: the robot is controlled by the goto or the avoid agent.

In conclusion, we can say that the hypothesis has been confirmed since the collaborative controller achieves the goal position faster and more precisely. Moreover, some unreachable states are achieved using collaboration.

5.2.2 Utility Function

This section is intended to prove that the distributed coordination mechanism based on the utility function is useful for dealing with conflicting resource usage.

In fact, the experiments performed in the previous section use the utility function to determine who wins the resource. Thus, Scenarios 2 and 3 are used to show the interaction between the goto and the avoid agents, as depicted in Figures 5.5 and 5.6. When the goto agent has control, it is represented by a continuous line, while when the avoid agent does, it is represented by a dotted one.

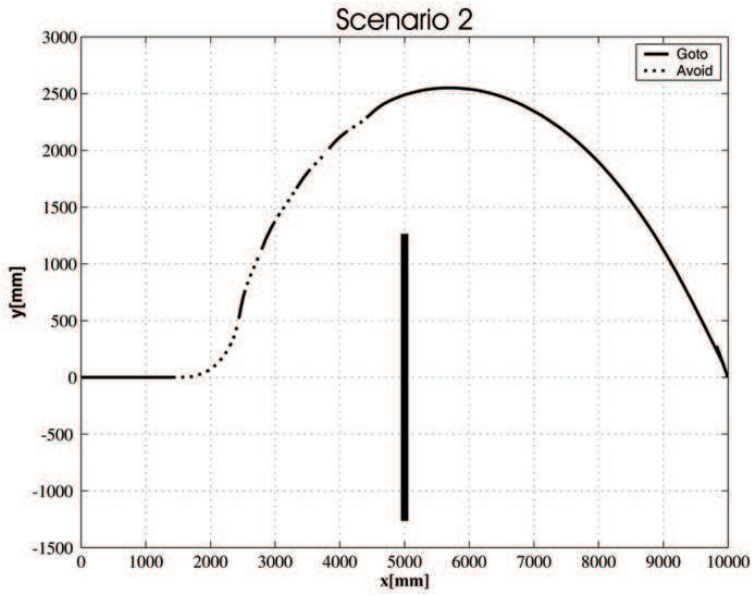


Figure 5.5: Interaction of the goto and avoid agents in Scenario 2

In Figure 5.5 the goto agent controls the robot agent (resource) until the obstacle is detected. Then, the avoid agent takes control of the resource until the object is no longer in the robot's trajectory. At this point, the goto agent recovers control and this interchange continues until the robot has completely got round the obstacle. Finally, the goto agent is in charge of guiding the robot to the goal point. It is important to note, that the trajectory depicted in this scenario corresponds to when the goto agent is using the collaborative controller. In cases where the goto agent uses the fast or the precise PID these interactions will repeat more often, depending on the speed of the robot and as explained in the previous section.

In Figure 5.6 the avoid agent starts controlling the robot agent because the sonar agent is detecting the wall at the side of the robot. It is necessary to recall that the avoid agent determines whether a detected obstacle is dangerous thanks to the creation of three zones around it, and takes appropriate action depending on which zone the robot is in (see subsection 3.5.2). But these zones increase in size according to the current speed of the robot, so depending on the experiment the obstacle is detected either sooner or later. In this particular case, the robot is inside the caution zone of the wall (the detected point is at 60 degrees more or less) and maintains its current heading, but its linear speed is limited. Then, as the robot approaches the column, it enters the danger zone and both angular and linear speeds are changed. When the obstacle is not detected then the goto agent wins the resource until the next obstacle is detected. These changes continue until the robot

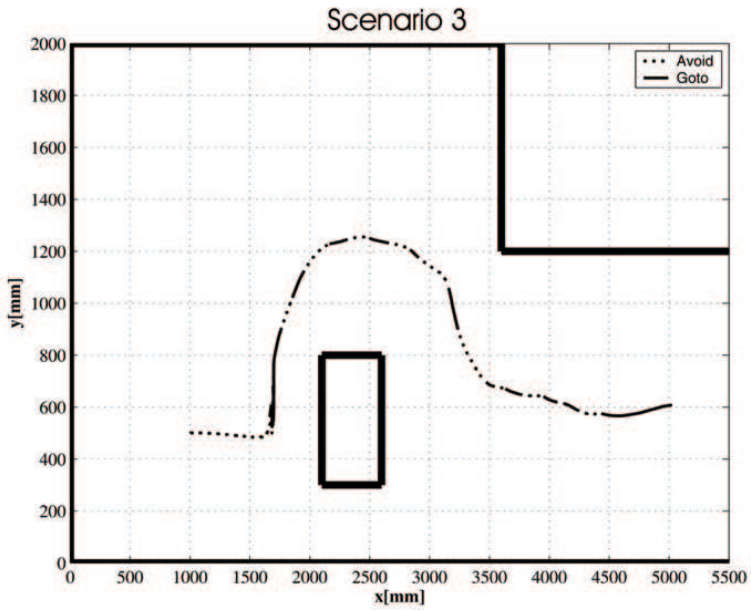


Figure 5.6: Interaction of the goto and avoid agents in Scenario 3

reaches the desired goal position.

It can be concluded from this experiment that agent interaction based on utility value allows the robot to perform complex trajectories with obstacles using a distributed coordination mechanism.

5.2.3 Resource Exchange

In this section the proposed fuzzy resource exchange method and two different ways of taking control of the resource are tested. The first occurs when the agents exchange control of the shared resource without using the proposed method, and the second when the proposed method is used. The hypothesis is that in the first case, a behavior will be obtained in which the robot moves abruptly while in the second case the behavior will be improved. Two main configurations are analyzed:

- **Abrupt:** changing control without using the resource exchange method.
- **Smoothing:** using our proposed method, i.e. using Equation 3.10 to effect a smoothing behavior when exchanging the resources after coordination.

In order to test the overall behavior of the architecture, the scenario shown in Figure 5.7 in which the robot must go from room A to room B is proposed. So that it can fulfill this task, the path planning agent provides the goto agent with the trajectory it needs to pass through the doors.

Figure 5.7 shows an example of the trajectory described by the robot when using the abrupt exchange of resources (grey line) and the proposed methodology (black line). As can be seen in this figure, at the points $(x, y) = (1.5, 3.3)m$ and $(x, y) = (6.3, 2.4)m$ approximately

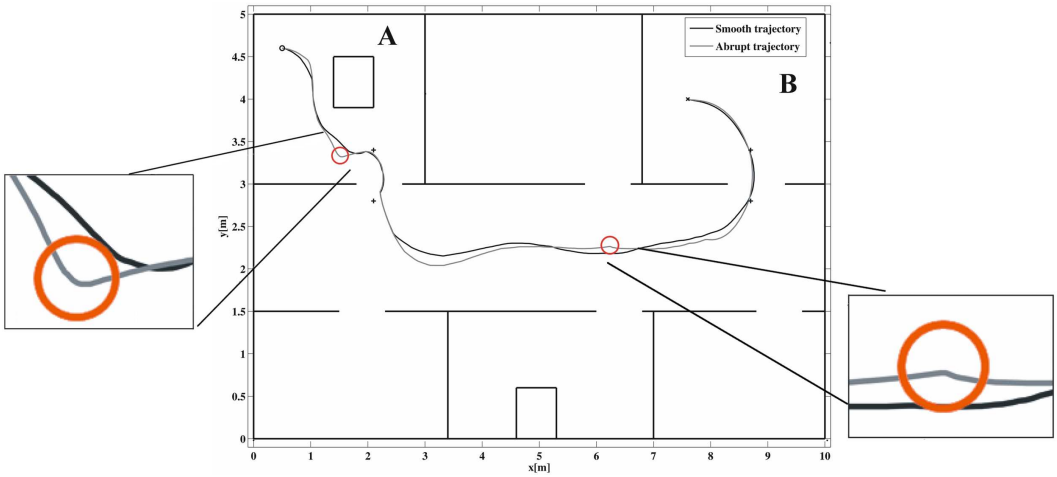


Figure 5.7: Example of a trajectory with our methodology.

(red circles), there are abrupt changes in the trajectory described by the robot when using the abrupt resource exchange methodology (grey line), but they are avoided when using the smoothing algorithm (black line).

The experiments consisted of five executions in the scenario for each configuration. Table 5.2 shows the average and standard deviation of each evaluation measure. Even though mean values are not so different between the two situations, deviations are enhanced.

Table 5.2: Comparison of the results of the two resource exchange methods.

Parameters	Abrupt	Smoothing
TD [m]	11.96 ± 0.103	11.99 ± 0.17
FO [°]	2.14 ± 0.47	2.16 ± 0.85
TT [s]	69.37 ± 2.43	64.39 ± 1.24
P [%]	99.648 ± 0.08	99.82 ± 0.05
TG [%]	73.86 ± 3.23	51.25 ± 5.41
TA [%]	26.14 ± 3.23	48.75 ± 5.41
TGt [%]	-	

Comparing the abrupt with the smoothing change of control, the most significant improvement is in the total time (**TT**) needed to reach the goal. In the *smoothing* configuration, **TT** has decreased, meaning that the cruising speed can be maintained for longer and changes of a short duration between agents are almost eliminated, thereby achieving softer trajectories. At the same time, the time the goto agent is in control (**TG**) has been decreased, while the avoid agent is in control in more consecutive cycles. Therefore, when an agent takes control of the resource, it does so for longer using the fuzzy resource exchange method than without using it. As a result, the agent can effectively follow its own goals for a longer period of time, and the robot's global behavior is consistently more coherent.

5.2.4 Agent Pattern

In this section agent design pattern is tested. The idea is to add a new agent to the control architecture and determine how difficult it is to implement first a new agent using the agent pattern and second the changes needed in the other agents, if they are necessary. In addition, the coordination mechanism is again tested since there is a new agent competing for the same resource.

Two scenarios have been defined. In both of them the robot must go from Room A to Room B while avoiding obstacles. In Scenario A there is a column in the middle of Room A that has been removed in Scenario B. To test the use of the design pattern two ARMADiCo configurations have been set up, according to two different development phases:

- **GA (Goto + Avoid):** only the goto and the avoid agents are controlling the robot
- **GAT (Goto + Avoid + GoThrough):** the gothrough agent has been added to test the difficulty of adding a new agent to the system as well as to verify that the emergent behavior of the overall system continues to be coherent and desirable.

Figures 5.8 and 5.9 depict both scenarios. Also, the trajectories described by the robot are shown in grey when only the goto and the avoid agent are competing for the robot agent and in dots when the gothrough agent is added.

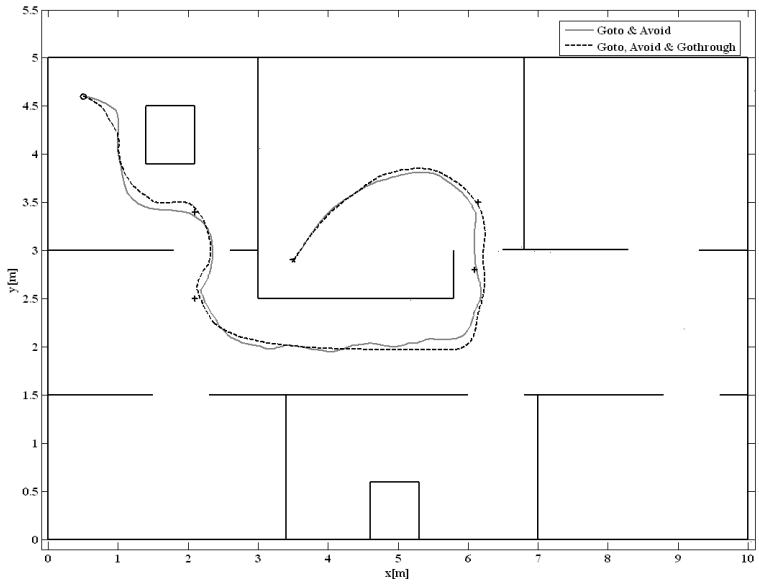


Figure 5.8: Scenario A to test the emergent behavior of the robot: goto + avoid agents.

Since the hypothesis is that the addition of the gothrough agent should improve the overall robot behavior, it is necessary to test how this happens. The experiments consisted of five executions in the scenario for each configuration. Table 5.3 shows the average and standard deviation of each evaluation measure.

Comparing the results for both scenarios, it can be seen that they are very similar, except for the total time (**TT**) needed to arrive at the destination. With the addition of the gothrough agent, this time is decreased, meaning that the average speed is higher than when there are only the goto and the avoid agents. Also, as expected, the time the avoid agent (**TA**) and the time of the goto agent (**TG**) intervenes decreases when the gothrough agent is added.

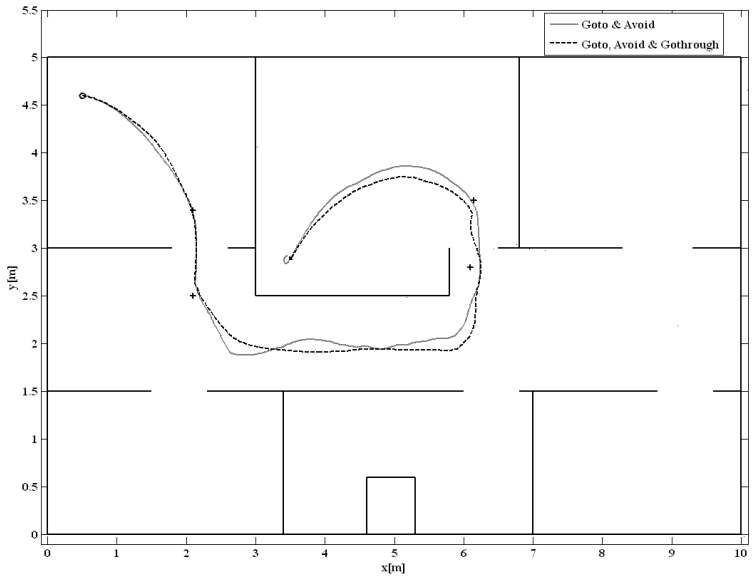


Figure 5.9: Scenario B to test the emergent behavior of the robot: goto + avoid + gothrough agents.

Table 5.3: Comparison of the results of the addition of a new agent to the control architecture.

Parameters	Scenario A		Scenario B	
	GA	GAT	GA	GAT
TD [m]	11.64 ± 0.14	11.84 ± 0.10	11.42 ± 0.22	11.55 ± 0.05
FO [°]	2.57 ± 0.60	1.56 ± 0.90	1.76 ± 1.08	1.93 ± 0.05
TT [s]	71.84 ± 6.85	60.17 ± 3.45	60.88 ± 5.21	51.14 ± 0.94
P [%]	99.37 ± 0.12	99.31 ± 0.21	99.33 ± 0.27	99.26 ± 0.20
TG [%]	68.64 ± 3.14	53.34 ± 2.13	73.16 ± 2.32	65.82 ± 1.73
TA [%]	31.36 ± 3.14	19.11 ± 1.74	26.83 ± 2.32	13.43 ± 0.29
TGt [%]	-	26.93 ± 2.65	-	20.16 ± 1.81

Perhaps the most important issue in this experiment is that introducing the gothrough agent is a simple matter since the agent pattern helps to determine the main aspects to consider. Also, no modifications have been performed on the existing agents because the requested services, the provided services and the share resources are defined when instantiating the agent. When the gothrough agent is added, the agents in the architecture, including the gothrough, establish the necessary links with the others to cover all the needs of the new agent and interact with it in case of conflicts. Because the resource determination algorithm is performed inside the agent and only involves the agent itself, no modifications have to be carried out on the others when a new agent is introduced (or deleted). Furthermore, the resource exchange method is carried out between two agents, the one that wins the resource and the one that had control of it; it is not important which the losing agent is, all that matters are the commands and the utility of the losing and winning agents to change

from the current state of the resource to the desired one. Thus, it must be concluded that following the agent design pattern no modifications to the existing agents have to be carried out.

5.2.5 Individual and Social Intelligences Integration

In order to demonstrate intelligence integration at the agent level and different agent interactions, the following experimental scenario is proposed: the robot is in Room A and its goal is to move to Room E (see Figure 5.10).

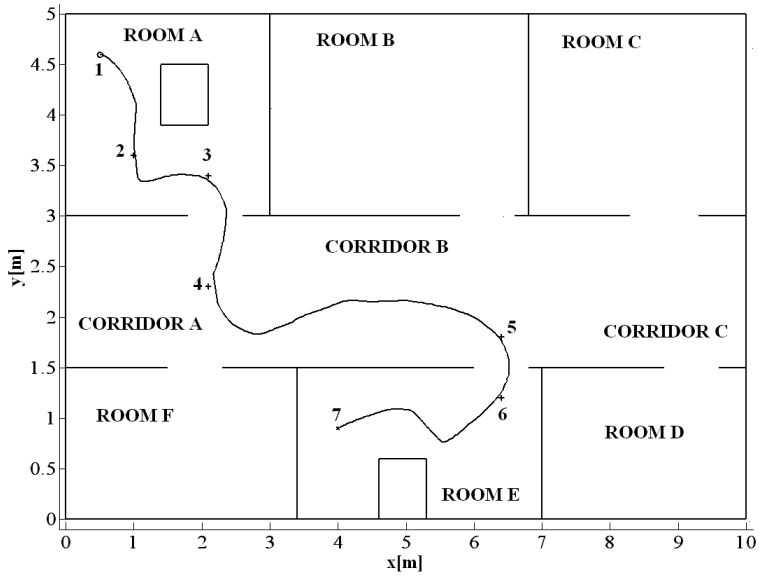


Figure 5.10: Robot task execution with ARMADiCo.

The agents being considered in this test are the following:

- Path planning agent:

  Individual intelligence: search method

  Social intelligence: -

- Goto agent:

  Individual intelligence: fuzzy collaborative controller

  Social intelligence: fuzzy-based smoothing method

- Avoid agent:

  Individual intelligence: PID controller

  Social intelligence: fuzzy-based smoothing method

- Sonar agent:

  Individual intelligence: probabilistic method

  Social intelligence: -

In this scenario, the following agent interactions occur. After receiving the mission from a human operator, the interface agent sends the desired destination (coordinates x_d,y_d and θ_d of Room E) from the path planning agent, and informs the encoder agent of the initial position (coordinates x_i,y_i,θ_i of Room A) on the global map. The path planning agent computes the trajectory needed to reach the destination and sends it to the goto agent, which at the same time receives information about the current position from the encoder agent. Concurrently, the avoid agent receives information from the sonar agent, and coordinates with the goto agent the speed commands to be sent to the robot agent. Finally, it is the robot agent that connects with the real robot to obtain the sensor readings and execute the speed commands. Figure 5.10 shows how this mission is achieved by the robot that follows a smooth trajectory, moving around the obstacles, going through doors, and moving through a corridor.

Figure 5.11 shows the interaction of the different intelligences integrated in the module-based architecture of the goto agent. At the beginning, the goto agent runs its individual fuzzy reasoning to achieve the different target points of the trajectory provided by the path planning. Concurrently, the coordination method with the avoid agent is also running: the goto agent is always the winner until this time. In the Goto/Avoid Coordination line can be seen the time dedicated to coordination. Then, at 40 seconds, the avoid agent takes control of the resource and as a result the goto agent dedicates most of its effort to getting control back. When it gets it, the goto agent uses its individual reasoning to achieve further points in the trajectory. This continues until such time as the robot has fulfilled its mission.

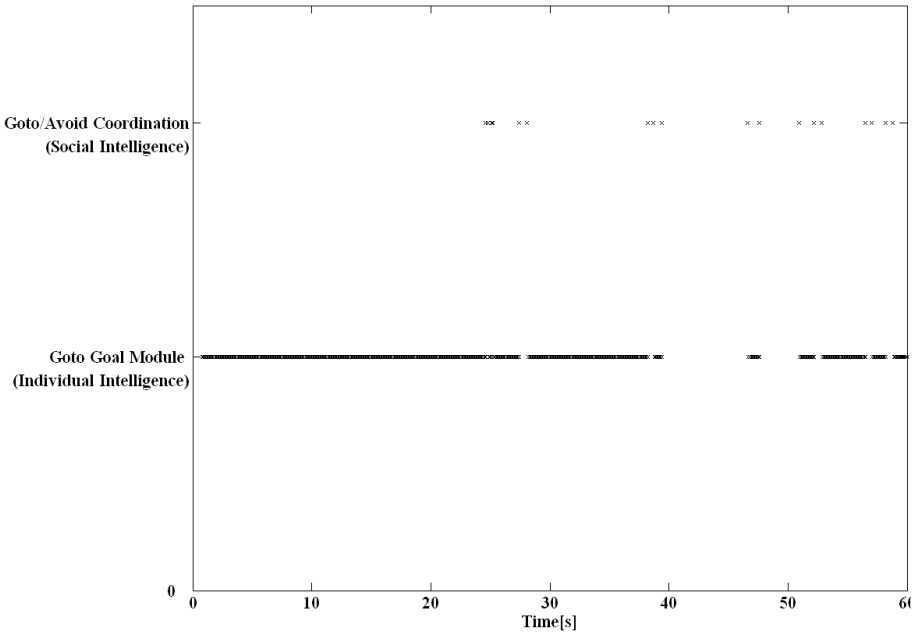


Figure 5.11: Module activation in the goto module-based agent.

In conclusion, it can be said that the integration of the individual and social intelligences proposed in the agent pattern helps the emergent behavior of the agent and the multi-agent system.

5.3 Real Robot

Finally, the control architecture is tested in a real environment. All the implemented agents are present in the system and the hypothesis is that the robot should perform a complex long trajectory.

Except for limiting the robot's maximum linear speed to $400mm/s$ because it slips on the floor of the building, no modifications to the control parameters (controllers and architecture) were carried out to test the robot in a real environment.

Two different experiments were performed with the robot.

5.3.1 Experiment 1

The first experiment consisted of it performing a short trajectory between two offices in the building. The robot had to move (with respect to the simulation values) from position $A = (0.49, 5.02, 90)$ to position $B = (4.69, 12.2, 180)$, passing through points $P_1 = (0.49, 5.82, 90)$, $P_2 = (2.89, 8.62, 90)$ and $P_3 = (3.49, 11.8, 0)$, each coordinate being (x, y, θ), x and y in meters and θ in degrees.

Figure 5.12: Part of the building.

Figure 5.12 shows the map of the two offices and the corridor between them. The other offices are not included on the map. All the doors are open, to identify their position on the map (only to clarify the reading of the map) but in fact in the experiments some of them were open and others closed. The offices in this building are quite small and very full of furniture, and would sometimes be impossible for the robot with its current sensoring information to find an exit. This is the reason why points A and B are very near the doors.

Figure 5.13 shows the trajectory and the ultrasound readings of the simulated experiment while Figure 5.14 depicts the real readings and trajectory.

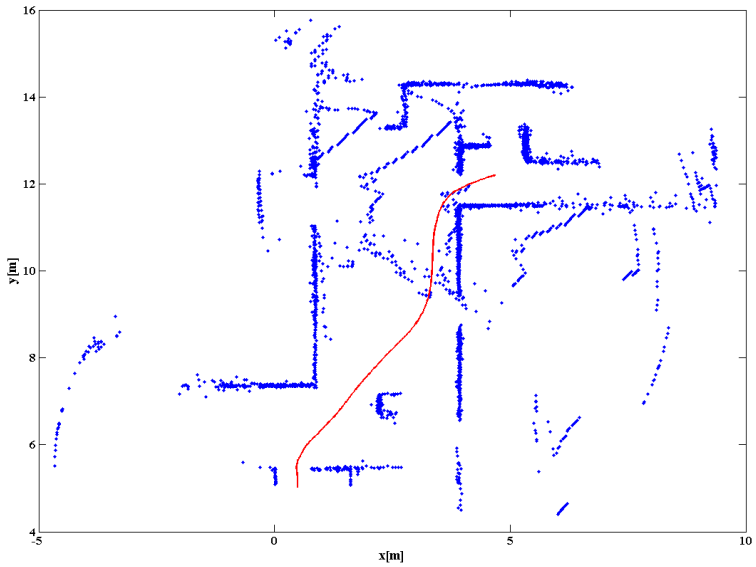


Figure 5.13: Sonar readings and trajectory described by robot during simulation.

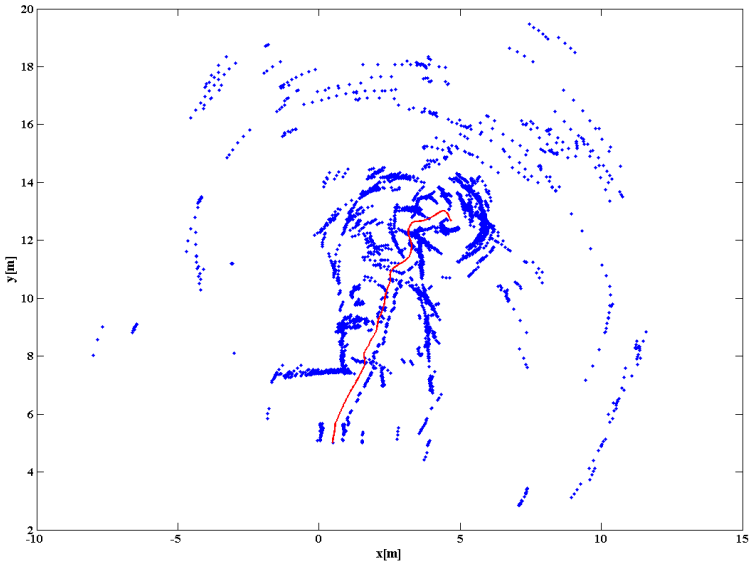


Figure 5.14: Sonar readings and trajectory described by robot during the real experiment.

Table 5.4 shows the values of the measurements defined at the beginning of this chapter, comparing the simulated and the real experiment.

Table 5.4: Results of the simulated and the real experiments.

Parameters	Simulated	Real
TD [m]	8.84	10.11
FO [°]	179.03	179.12
TT [s]	64.8	65
P [%]	99.89	99.84
TG [%]	78.43	67.59
TA [%]	8.78	12.59
TGt [%]	12.79	19.66

5.3.2 Experiment 2

The second experiment was to allow the robot to run a very long distance, from one side of the building to the other, and have it detect obstacles such as people and columns, move inside corridors, detect open spaces such as the stairhead and deal with non reflecting walls.

The floor map of the building is shown in Figure 5.15. Rooms and doors are not shown on this map, but only the corridor through which the robot had to move. During the real experiments, some of them were open and others closed. The robot had to go from position $A = (1.15, 1.2, 0)$ to position $B = (5.8, 38.1, 180)$, passing through points $P_1 = (5.6, 0.8, 0)$ and $P_2 = (13.15, 6.8, 90)$, each coordinate being (x, y, θ), x and y in meters and θ in degrees.

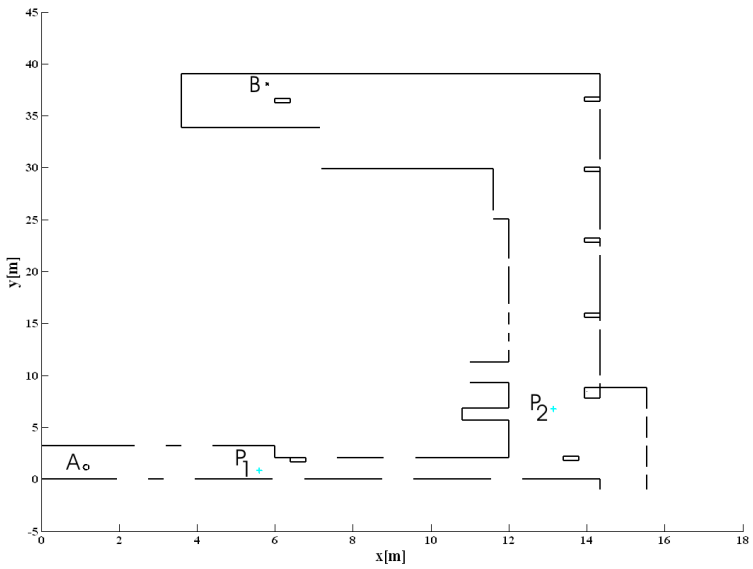


Figure 5.15: Floor of the building.

In Figure 5.16 the simulated trajectory described by the robot to reach the desired position is shown. Also, the sonar readings along the course of the movement are depicted.

Figure 5.17 shows the real trajectory described by the robot. Table 5.5 shows the results of

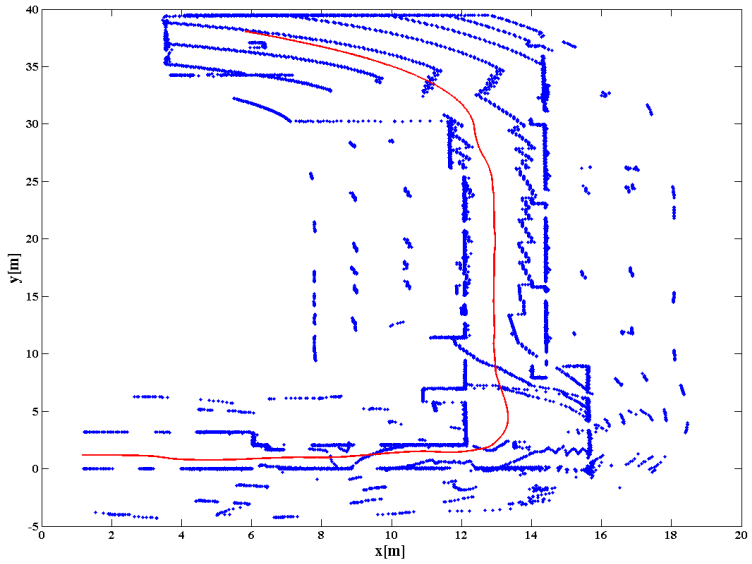


Figure 5.16: Sonar readings and trajectory described by robot during simulation.

the experiment. These results of the simulated and real experiments cannot be compared since in the real experiment the robot did not reach the physical goal destination, although it believed that it was in this position. An important issue to highlight is that in the real experiment the avoid agent had control of the resource more times than in the simulation, but the gothrough agent did not.

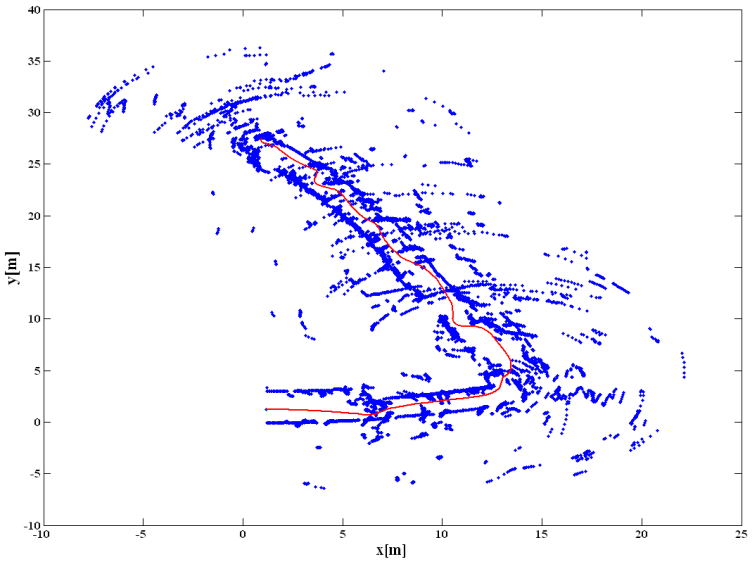


Figure 5.17: Sonar readings and trajectory described by robot during the real experiment.

As can be seen in Figure 5.17 the encoder readings accumulate errors that affect the position of the robot as well as its heading, causing the robot to believe it is in a position that is not the real one. These errors, which are not produced by the control architecture but are a result of the sensorization of the robot, could be minimized by using the localization agent or performing SLAM (self localization and mapping).

Table 5.5: Results of the simulated and the real experiments.

Parameters	Simulated	Real
TD [m]	51	41.67
FO [°]	178.95	76.73
TT [s]	152.71	142.8
P [%]	99.94	68.86
TG [%]	83.24	59.55
TA [%]	7.72	31.49
TGt [%]	9.03	8.96

5.4 Conclusions

In this chapter some simulated and real results have been presented with a view to demonstrating the advantages and finding the limitations of the proposed architecture.

First, the performance of the collaborative controller developed inside the goto agent was tested when coordination with other agents in the architecture was needed. The results obtained showed that the collaborative controller performed better than the simple controllers even though interacting with other agents.

Second, the utility function that serves to implement the distributed coordination was tested. This experiment is related to the first one, because the utility function was already in use. Results showed that utility values can be used to distribute coordination.

Third, the proposed resource exchange method was tested. It improved the resulting trajectory, and short period interactions between the coordinated agents were eliminated, with each agent achieving control of the resource for a longer time. To test the fuzzy method, two ways of changing the control of the resource were compared.

Fourth, a new behavioral agent was added to the community to test the agent design pattern and how the new agent affected the coordination mechanism. It was proven that with the agent pattern design whether or not an agent joined or left the community did not affect the other agents, but that overall behavior could be improved depending on the agent's local behavior. It was also proven that the coordination mechanism continued working even though more agents were sharing the same resource.

Fifth, intelligence integration at the agent level and in the multi-agent control architecture were tested. As individual and social intelligences use their own AI technique to obtain the goal they are pursuing inside the agent, it was important to analyze the interactions between these two methods. Additionally, as each agent has its own AI technique to perform its task, it was necessary to see if the overall behavior of the control architecture was coherent and desired. All these features were proven to work correctly in the proposed architecture.

Finally, the whole architecture was tested in a real environment. Even though the mission was a very simple one, the trajectory described by the robot was complex since the total distance traveled was long enough to cause errors in the sensors' readings and inaccuracies in the mental states of the agents.

As a conclusion to all of the experiments carried out it can be said that ARMADiCo fulfils

most of its design requirements, namely a multi-agent architecture with distributed coordination able to control a real robot in real time.

The challenge of building such a multi-agent architecture lies in the extra communication requirements that the module-based architectures do not have.

The distributed coordination and the modular organization of the agents (social and individual intelligences), which result in a coherent emergent behavior of the overall approach, is the most significant achievement, taking into account that everything is performed in real time.

However, additional experiments should be considered in the future, in order to establish the suitability of ARMADiCo for dealing with complex tasks. For example:

- The incorporation of new agents, particularly deliberative agents.
- The analysis of resource latency when exchanging such agents.

Also, some limitations were detected in the real experiments, such as the error in the position and heading of the robot. As pointed out, they could be overcome with the incorporation of a localization agent or SLAM agent.

CHAPTER 6

Conclusion and Future Work

The unknown future rolls toward us. I face it, for the first time, with a sense of hope. Because if a machine, a Terminator, can learn the value of human life, maybe we can too.

From *Terminator 2: The Judgement Day* by James Cameron

This chapter concludes the work developed in this dissertation. First, the conclusions of the thesis are presented by reviewing the contents described in the different chapters. Next, the contributions extracted from the proposal and the results are shown. Publications resulting from this work are then listed and finally, some interesting research issues for the future are indicated in the future work section.

6.1 CONCLUSIONS

This Ph.D. thesis deals with an architecture for controlling a mobile robot in accordance with the main objective in robotics, which is to develop a robot control system capable of intelligent and suitable responses to changing environments. The control architecture has to possess a number of desirable features: flexibility, real-time response, coherent behavior, adaptability, fault tolerance, scalability, easy design and granularity.

Chapter 2 presents the three paradigms used to develop robot control architectures, the reactive, the deliberative and the hybrid paradigms, highlighting the advantages and the disadvantages of each of them, as well as those that are the most representative. The hybrid paradigm is the most used since it combines the other two. Several of these are described: first modular architectures; then multi-agent architectures used to control a single robot; next multi-agent approaches to control multi-robot systems; then robotic development environments; and finally, the architectures developed at the University of Girona. From all the architectures analyzed, several properties are extracted and used to compare all the architectures. The desirable features of the proposed architecture, ARMADiCo, are then identified.

Chapter 3 describes the proposed architecture: a multi-agent control architecture for a single robot with distributed coordination. To develop the architecture, an engineering methodology, MaSE, is followed to engineer and verify a coherent design. Furthermore, an agent design pattern is defined and used to develop the agents that form the multi-agent system. This pattern simplifies the incorporation of new agents and captures their common features. The implementation of the pattern is done through a modular architecture, since it guarantees the different lines of reasoning of the agents. In order to implement the agents

of the architecture several Artificial Intelligence Techniques are used in accordance with the required reasoning level. These techniques can be found in the individual intelligence of the agent, the method through which the agent achieves its own goals; and in the social intelligence of the agent, the method used to interact with and manage other agents in the architecture. With these two kinds of intelligences an emergent behavior is obtained both at the agent and at the multi-agent level. In addition, a distributed coordination method is proposed, based on a utility calculation. With this method, each agent is capable of determining which of the competing agents will get control of the resource. Moreover, once the agent gets resource control, a fuzzy based method is used to produce a smooth change in the control of the resource, avoiding unexpected behaviors in the robot.

Chapter 4 outlines the MaSE methodology, the multi-agent platform developed to run the control architecture, the robot platform, the dynamic model of the robot and the collaborative controller required to move the robot. In the case of the MaSE methodology, all its steps are explained and a number if diagrams are shown. With regard to the multi-agent platform, a commercial platform is explained first, but at the end, the ad-hoc platform developed for the robot architecture is detailed. This platform allows agents to be executed at different rates, including real-time, and to be run on different computers. In addition, the platform allows coding in different programming languages if the communication protocol and language among the agents is maintained. With regard to the robot platform, the Pioneer 2DX robot is described in order that the available sensory information and actuators and the way data is accessed may be known, because they constrain the manner that agents must interact with the real world. Also, the mathematical equations and the values of the different parameters that describe the dynamic model have been empirically found. This model has been used to create a 3D simulator in Simulink/Matlab$^{(R)}$ to develop the controllers of the behavioral agents. Finally, the collaborative controller is presented. This controller uses a fuzzy method to combine the response of two separate PID's, one fast but not precise and the other precise but slow. Also, a number of responses of the three controllers are depicted, showing that better results are obtained with the collaborative controller, both in time, in precision and in reaching some states that, due to the non-linear nature of the robot, are not possible with the separate controllers.

Finally, Chapter 5 outlines several experiments carried out using the simulator and the real robot. Experiments were designed to test the collaborative controller in the MAS environment; to analyze how well the utility function performs; to try out the resource exchange, checking that undesirable responses are not obtained; to verify the feasibility of the agent pattern when creating and adding a new agent, confirming that no changes to the other agents are needed; to analyze individual and social intelligences integration; and finally to test the behavior of the real robot when asked to carry out a relevant mission.

6.2 CONTRIBUTIONS

The main goal of this work was to develop a robot control architecture for a mobile robot that supports complex robot behaviors. This goal has been accomplished and in the process some research contributions achieved. These contributions are:

- **Multi-agent Control Architecture:** the most important contribution of this thesis is the multi-agent architecture with distributed coordination for controlling an autonomous robot. The fact that the components of the architecture are agents allows their parallel execution at different rates (real-time, if necessary). Furthermore, as each agent specializes at solving a particular problem, when new hardware is added new agents to manage it can join the community and create the links necessary to interact with other agents. Also, when new software capabilities are needed, agents with the appropriate Artificial Intelligence technique can be incorporated into the control architecture. This

allows a flexible architecture in which components can be added to adapt the robot to different environments using the proper set of agents necessary for dealing with the new situations. Finally, the agents in the community can be spread over several computers and coded in different programming languages, thereby permitting a more flexible architecture, as well.

- **Distributed Coordination Mechanism:** in the architecture, coordination is distributed among the agents that share the resource. This coordination is achieved using the utility values. It requires a message to be passed among the agents that share the resource, but communication overhead is reduced limiting the agent that has control to sending the utility value to the others, until another agent has a greater utility value.

- **Resource Exchange:** the proposed coordination method takes into account that abrupt changes in the resource should be avoided. A fuzzy system is proposed to produce change in a smooth manner. The fuzzy system determines a time window frame and uses it along with the utility value and the control commands to produce a smooth change from the current control commands to those of the winning agent in a maximum time specified by the time window frame. With this resource exchange method, disruptions to the resulting trajectory of the robot are avoided.

- **Pattern Design:** a pattern design is used as a methodological approach for designing agents in the distributed environment. It allows common features of all the agents to be defined and simplifies the task of creating new ones. This pattern helps in the design of the agent's goals and the way to accomplish them; how the internal state of the agent is maintained; the resources needed and thus the agents to compete with; the way of determining who wins the resource and how the change from one state of the resource to the other is done; and the agents that provide the required services, i.e. the agents to collaborate with. This pattern has also allowed module based agents to be defined. Inside the agent two modules can be differentiated from the others since they allow the agent to pursue their goals (individual intelligence) and the coordination and management of resources with other agents (social intelligence). Separation of individual and social intelligences permits the use of different AI techniques to achieve them separately and to define the interactions among the agents in the architecture. This produces an emergent behavior at both the agent and at the multi-agent level.

- **Collaborative Control:** The definition of the collaborative controller has helped to mix the outputs of PID controllers intended to separately achieve particular specifications that are sometimes opposed. With this controller, a better result in the movement of the robot has been achieved, as much in the time required as in the way the trajectory is described. Also, it is possible to reach some states that could not have been obtained with separate controllers.

- **Multi-agent Platform:** It was necessary to implement a multi-agent platform to achieve some of the desirable capabilities for the architecture, such as the real-time where there is no agent that centralizes the architecture. Besides having a basic agent that implements most of the functions of the agent (communication, message treating, registering, etc.), this platform has several agents like the DF, the wakeup and the monitor agents that implement the yellow pages and the management services established by FIPA standards. These agents are meant to detect and repair faults in the software platform.

6.3 Publications

This section is dedicated to presenting the publications connected with the research project developed in the last years. First, publications in books, international journals and conferences obtained from the results of the Ph.D. thesis are enumerated according to the year of publication. Next, there is a list of the publications that are partially related to the thesis because they are outputs of my contribution to other projects undertaken in the research group.

6.3.1 Thesis Publications

Book Publications

Physical agents, Julián, V.; Carrascosa, C.; Corchado, J.M.; Bajo, J.; del Acebo, E.; Innocenti, B. and Botti, V. *Issues in Multi-Agent Systems. The AgentCities.ES Experience*, Moreno, Antonio; Pavón, Juan (Eds.): Whitestein Series in Software. Agent Technologies and Autonomic Computing. A Birkhäuser book, Springer-Verlag, pp. 117 – 143, *ISBN:* 978-3-7643-8542-2, 2008.

International Journal publications

A multi-agent architecture with cooperative fuzzy control for a mobile robot, Innocenti, B.; López, B. and Salvi, J., **Robotics and Autonomous Systems**, Vol. 55(12), pp. 881 – 891,*ISSN:* 0921-8890, 2008.

Multi-agent planning architecture for autonomous robots, Innocenti, B.; López, B. and Salvi, J., **The Planet Newsletter** (7), pp. 24 – 27, http://planet.dfki.de/service/Newsletter/ index.html, *ISSN:* 1610-0204, 2003.

Conference Contributions

Design patterns for combining social and individual intelligences on modular-based agents, Innocenti, B.; López, B. and Salvi, J., **3rd International Workshop on Hybrid Artificial Intelligence Systems (HAIS)**, Hybrid Artificial Intelligence Systems, Third International Workshop, Lecture Notes in Computer Science, Volume 5271/2008, Springer Berlin / Heidelberg, pp. 70-77, *ISBN: 978-3-540-87655-7*, Burgos, Spain, September 24th - 26th, 2008.

Integrating individual and social intelligence into module-based agents without central coordinator, Innocenti, B.; López, B. and Salvi, J., **4th European Starting AI Researcher Symposium (STAIRS)**, Stairs 2008: Proceedings of the Fourth Starting AI Researchers' Symposium, Frontiers in Artificial Intelligence and Applications, Amadeo Cesta and Nikos Fakotakis, editors, IOS Press, pp. 82 – 94, *ISBN*:978-1-58603-893-9, Patras, Greece, July 21st - 25th, 2008.

Resource coordination deployment for physical agents, Innocenti, B.; López, B. and Salvi,

J., **6th Int. Workshop of AAMAS: From Agent Theory to Agent Implementation**, pp. 101 – 108, Estoril, Portugal, May 12th - 13th, 2008.

A multi-agent system with distributed coordination for controlling a single robot, Innocenti, B.; López, B. and Salvi, J., **7th Portuguese Conference on Automatic Control (CONTROLO)**, Proceedings of 7th Portuguese Conference on Automatic Control (CDRom), Lisbon, Portugal, September 11th - 13 th, 2006.

A multi-agent collaborative control architecture with fuzzy adjustment for a mobile robot, Innocenti, B.; López, B. and Salvi, J., **3rd International Conference on Informatics in Control, Automation and Robotics (ICINCO)**, Proceedings of Third International Conference on Informatics in Control, Automation and Robotics, pp. 523–526, *ISBN:* 972-8865-60-0, Setúbal, Portugal, August 1st - 5th, 2006.

How MAS support distributed robot control, Innocenti, B.; López, B. and Salvi, J., **37th International Symposium on Robotics (ISR)**, Proceedings of 37th International Symposium on Robotics (ISR), *ISBN:* 3-18-091956-6, Munich, Germany, May 15th - 17th, 2006.

Una arquitectura multi-agente con control difuso colaborativo para un robot móvil, Innocenti, B.; López, B. and Salvi, J., **Campus Multidisciplinar en Percepción e Inteligencia (CMPI). Una perspectiva de la Inteligencia Artificial en su 50 Aniversario**, Ed. Antonio Fernandez-Caballero, Maria Gracia Manzano, Enrique Alonso, Sergio Miguel Torné. Vol.1., pp. 413–424, *ISBN:* 84-689-9560-6, Albacete, Spain, July 10th - 14th, 2006.

Dynamical Model Parameters Identification of a Wheeled Mobile Robot, Innocenti, B.; Ridao, P.; Gascons, N.; El-Fakdi, A.; López, B. and Salvi, J., **5th IFAC Symposium on Intelligent Autonomous Vehicles**, Proceedings of 5th IFAC Symposium on Intelligent Autonomous Vehicles, Lisbon, Portugal, July 5th - 7th, 2004.

Multi-agent system architecture with planning for a mobile robot, Innocenti, B.; López, B. and Salvi, J., **X Conferencia de la Asociación Española para la Inteligencia Artificial (CAEPIA)**, CAEPIA 2003, Donostia, Spain, November 12th - 14th, 2003.

6.3.2 Other Related Publications

International Journal Publications

A Multiagent System for Coordinating Ambulances for Emergency Medical Services, López, B.; Innocenti, B.; Busquets, D., **IEEE Intelligent Systems**, Volume 23, Issue 5, Sept.-Oct. 2008, pp. 50 – 57, 2008.

Conference Contributions

Coordinación del transporte sanitario basado en subastas, Innocenti, B.; Busquests, D.; López, B., **XI Congreso Nacional de Informática Médica (INFORMED 2006)**, Libro de comunicaciones, ISBN: 690-0102-7, Múrcia, Spain, November 14th - 16th, 2006.

How multi-agent systems support acute stroke emergency treatment, López, B.; Aciar, S. ; Innocenti, B.; Cuevas, I., **3rd Workshop on Agents Applied in Health Care (IJCAI**

2005), Proceedings of the 3rd Workshop on Agents Applied in Health Care (IJCAI 2005), Edimburg, England, July 30th - August 5th, 2005.

A multi-agent system to support ambulance coordination in time-critical patient treatment, López, B.; Innocenti, B.; Aciar, S.; Cuevas, I., **VII Simposio Argentino de Inteligencia Artificial (ASAI 05)**, CD Rom, ISBN: 1666 1079, Rosario, Argentina, August 29th September 2nd, 2005.

Estudio preliminar sobre la interacción de agentes físicos heterogéneos, Ramón, J.A.; de la Rosa, J.Ll.; Innocenti, B., **3rd Workshop on Physical Agents (WAF'02**), Murcia, Spain, March 14th - 15th, 2002.

RogiTeam description, de la Rosa, J.Ll.; Innocenti, B.; Montaner, M.; Figueras, A.; Muñoz, I.; Ramón, J.A., **RoboCup 2001: Robot Soccer World Cup V**, Lecture Notes in Artificial Intelligence, Vol. 2377, Ed. A. Birk. S. Codareschi, S. Tadokoro, pp: 587–590, ISBN: 3-540-43912-9, Seattle, United States, August 2nd - 10th, 2002.

Examples of dynamical physical agents for multi robot control, Innocenti, B.; de la Rosa, J.Ll.; Vehí, J.; Muñoz, I.; Montaner, M.; Fàbregas, M.; Figueras, A.; Ramón, J.A., **2nd Workshop Hispano-Luso de Agentes Físicos**, Madrid, Spain, March, 2001.

Preliminary studies of dynamics of physical agents ecosystems, de la Rosa, J.Ll.; Muñoz, I.; Innocenti, B.; Figueras Coma, A.; Montaner, M.; Ramón, J.A., **RoboCup 2000: Robot Soccer. World Cup IV**, Lecture Notes in Artificial Intelligence, Vol. 2019, Ed.P. Stone, T. Balch, G. Kraetzschmar, pp: 373–378, ISBN: 3-540-42185-8, ISSN: 0302-9743, Melbourne, Australia, August 26th - September 3rd, 2001.

RoGiTeam description, de la Rosa, J.Ll.; Innocenti, B.; Montaner, M.; Figueras, A.; Muñoz, I.; Ramón, J.A., **RoboCup 2000: Robot Soccer. World Cup IV**, Lecture Notes in Artificial Intelligence, Vol. 2019, Ed.P. Stone, T. Balch, G. Kraetzschmar, pp: 551–554, ISBN: 3-540-42185-8, ISSN: 0302-9743, Melbourne, Australia, August 26th - September 3rd, 2001.

Rogi 2 team description, de la Rosa, J.Ll.; García, R.; Innocenti, B.; Muñoz, I.; Figueras Coma, A.; Ramón, J.A., **RoboCup-99: Robot Soccer World Cup III**, Lecture Notes in Artificial Intelligence, Vol. 1856, Ed.M. Veloso, E. Pagello, H. Kitano, pp: 679–682, ISBN: 3-540-41043-0, Stockholm, Sweden, July 31st - August 6th, 2000.

Rational Agents by Reviewing Techniques, de la Rosa, J.Ll.; Innocenti, B.; Muñoz, I.; Montaner, M., **16th International Joint Conference on Artificial Intelligence, IJCAI. RoboCup-99: Robot Soccer World Cup III**, Lecture Notes in Artificial Intelligence Vol. 1856, Ed.M. Veloso, E. Pagello, H. Kitano, pp: 632 - 637, ISBN: 3-540-41043-0, Stockholm, Sweden, July 31st - August 6th, 2000.

Rogi Team Real: Dynamical Physical Agents, de la Rosa, J.LL.; García, R.; Innocenti, B.; Muñoz, I.; Figueras, A.; Ramón, J.A.; Montaner, M., **16th International Joint Conference on Artificial Intelligence, IJCAI, RoboCup-99: Robot Soccer World Cup III**, Lecture Notes in Artificial Intelligence Vol. 1856, Ed.M. Veloso, E. Pagello, H. Kitano, pp: 434–438, ISBN: 3-540-41043-0, Stockholm, Sweden, July 31st - August 6th, 2000.

El sistema integrado de la plataforma de experimentación de robots móviles, autónomos y cooperantes, de la Rosa, J.Ll.; Innocenti, B., **Primer Congreso Internacional de Ingenierías Eléctricas y Electrónicas**, Bucaramanga, Colombia, March, 2000.

An example of dynamical physical agents, de la Rosa, J.Ll.; Innocenti, B.; Oller, A.; Figueras, A.; Ramón, J.A.; Muñoz, I.; Montaner, M., **2nd EuroRoboCup Workshop**, Amsterdam, Netherlands, June, 2000.

Uso de parámetros dinámicos para mejorar la cooperación entre robots móviles. Aplicación en un convoy de dos robots., Oller, A.; de la Rosa, J.Ll.; Innocenti, B., **Seminario Anual de Automática, Electrónica Industrial e Instrumentación. Técnicas de Agentes (SAAEI'99)**, Seminario Anual de Automática, Electrónica Industrial e Instrumentación, pp: 311–314, ISBN: 84-699-0923-1, Madrid, Spain, 1999.

6.4 Future Work

The development of this work has led to new problems and topics that need further research. Some of them are listed below in a hypothetical chronological order:

- **Adding new agents to the multi-agent architecture:** the most important future step is to add more agents, both behavioral and deliberative, to the architecture. With behavioral agents, adding new agents will permit the distributed coordination mechanism and the resource exchange method to be thoroughly tested. As far as deliberative agents are concerned, the current implementation of the path planning agent must be improved, and the mission planning agent and the localization agent must be added to test more complex tasks.
- **Testing a coordination mechanism for deliberative agents:** the first step will lead to testing the trajectory merging method proposed to coordinate deliberative agents, since in this work only the one proposed for behavioral agents has been tested. Further analysis is also required for the mission planning abstract agent that in turn is itself a multi-agent system.
- **Comparing ARMADiCo with other architectures:** to compare several robot control architectures, a set of measurement indexes needs to be defined. These indexes should consider the most relevant features of the architectures and must allow their comparison not only in a qualitative but also in a quantitative way. In this work a qualitative comparison has been made, but it will be necessary to perform the quantitative measurements to establish a formal criterion to determine how good the control architectures and the strong points of each one are. With this set of measurements it will be possible to rank the existing control architectures.
- **Adding learning capabilities:** in order to make ARMADiCo suitable for new environments, learning mechanisms will have to be incorporated. For example, learning mechanisms in behavioral agents should allow the parameters of the agents to adapt to new situations and new hardware platforms. Reinforcement learning seems a good option to consider, since it gives good results in other robotic systems. Regarding the wakeup agent, the learning mechanism should allow the appropriate agents for a given situation to be chosen automatically, starting with the agents that are suitable for it. For example, in outdoor missions, the wakeup agent should choose the GPS sensor agent to localize the robot instead of doing it with the encoders and sonar agents, as it should do for indoor missions.
- **Testing ARMADiCo in other robots:** the next step is to test the proposed architecture in other mobile platforms. This will allow the minimum changes that must be performed in the agents to adapt the architecture to the new hardware platform to be set. We can at least foresee changes to the robot agent (the interface between the multi-agent system and the real robot) and some parameters of the controllers of the behavioral agents. Whether changes will include deliberative agents or the coordination mechanisms will have to be analyzed. Once tested on mobile robots, it would seem worthwhile to try ARMADiCo in other kind of robots.
- **Improving the collaborative controller:** more controllers should be defined in order to improve the collaborative controller of the goto agent. Also, the merging of

the control commands may include parameters other than distance, as for example consumed energy, time to reach the goal, etc. Changing controllers while executing a task is still an open issue in the control field. To test the collaborative controller a comparison with more elaborated controllers should also be carried out.

- **Trying out ARMADiCo with multi-robot systems:** another interesting aspect of the proposed architecture to cover is its testing in a multi-robot environment. This will allow the interactions of the multi-agent systems of each robot to be analyzed, and provides a good perspective for distributing the planning of the multi-robot system's mission in the robots.

Bibliography

[1] V. Andronache and M. Scheutz. ADE: A tool for the development of distributed architectures for virtual and robotic agents. *Proceedings of the 4th International Symposium From Agent Theory to Agent Implementation*, pages 606 – 611, 2004.

[2] R. C. Arkin. *Towards Cosmopolitan Robots: Intelligent Navigation in Extended Man-Made Environments*. PhD thesis, COINS Technical Report 87-80, Department of Computer and Information Science, University of Massachusetts, 1987.

[3] R. C. Arkin. *Behavior-Based Robotics*. The MIT Press, 1998.

[4] H. Asama, A. Matsumoto, and Y. Ishida. Design of an autonomous and distributed robot system: Actress. *Proceedings. IEEE/RSJ International Workshop on Intelligent Robots and Systems '89. (IROS '89) 'The Autonomous Mobile Robots and Its Applications'*, pages 283 – 290, 1989.

[5] T. Balch. Teambots, 2004. http://www.teambots.org/.

[6] T. Balch and A. Ram. Integrating robotics research with JavaBots. *Working Notes of the AAAI 1998 Spring Symposium*, 1998.

[7] G. A. Bekey. *Autonomous Robots: From Biological Inspiration to Implementation and Control*. The MIT Press, Cambridge, Massachussetts, London, England, 2005.

[8] D. Blank, D. Kumar, L. Meeden, and H. Yanco. Pyro: A python-based versatile programming environment for teaching robotics. *Journal on Educational Resources in Computing (JERIC).*, 4(3):1 – 15, 2004. Special issue on robotics in undergraduate education. Part 2.

[9] D. Blank, D. Kumar, L. Meeden, and H. Yanco. The pyro toolkit for AI and robotics. *AI Magazine*, 27(1):39 – 50, 2006.

[10] R. P. Bonasso, R. J. Firby, E. Gat, D. Kortenkamp, D. Miller, and M. Slack. Experiences with an architecture for intelligent, reactive agents. *Journal of Experimental and Theoretical Artificial Intelligence*, 9(2/3):237–256, 1997.

[11] R.P. Bonasso, R. Kerr, K. Jenks, and G. Johnson*. Using the 3t architecture for tracking shuttle RMS procedures. *Proceedings of the IEEE International Joint Symposia on Intelligence and Systems*, pages 180 – 187, 1998.

[12] V. Botti, C. Carrascosa, V. Julian, and J. Soler. Modelling agents in hard real-time environments. *Lecture Notes in Computer Science*, Volume 1647/1999:63 – 76, 1999.

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

[14] H. Bruyninckx. Open robot control software: The OROCOS project. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 2001)*, 3:2523–2528, 2001.

[15] D. Busquets, C. Sierra, and R. López de Màntaras. A multiagent approach to qualitative landmark-based navigation. *Autonomous Robots*, 15:129 – 154, 2003.

[16] G. Campion, B. D'Andrea-Novel, and G. Bastin. Modelling and state feedback control of Non Holonomic mechanical systems. *Proceedings of the 30th Conference on Decision and Control. Brighton, England*, pages 1184 – 1189, 1991.

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

[18] C. Carrascosa, J. Bajo, V. Julian, J.M. Corchado, and V. Botti. Hybrid multi-agent architecture as a real-time problem-solving model. *Expert Systems with Applications*, 34:2–17, 2008.

[19] B. Chaib-Draa, B. Moulin, and I. Jarras. Systèmes multi-agents : Principes généraux et applications. *Principes et architecture des systèmes multi-agents, JP. Briot et Y Demazeau (eds) (Hermes, Lavoisier)*, 2001.

[20] L. Chaimowicz, A. Cowley, V. Sabella, and C. J. Taylor. ROCI: A distributed framework for multi-robot perception and control. *Proceedings of the 2003 IEEE/RSJ Intl. Conference on Intelligent Robots and Systems*, pages 266 – 271, 2003.

[21] S.P. Chan. An efficient algorithm for identification of robot parameters including drive characteristics. *Journal of Intelligent and Robotic Systems*, 32(3):291 – 305, 2001.

[22] W. Chung, G. Kim, and M. Kim. Development of the multi-functional indoor service robot PRS systems. *Autonomous Robots*, 22:1–17, 2007.

[23] CLARAty. on-line: www.claraty.jpl.nasa.gov/.

[24] J.H. Connell. SSS: A hybrid architecture applied to robot navigation. *Proceedings of the IEEE International Conference on Robotics and Automation*, 3:2719 – 2724, 1992.

[25] E. Coste-Maniere and R. Simmons. Architecture, the backbone of robotic systems. *IEEE International Conference on Robotics and Automation, 2000. Proceedings. ICRA '00.*, 1:67 – 72, 2000.

[26] C. Côté. Mobile and autonomous robotics integration environment (MARIE), 2005. http://marie.sourceforge.net/.

[27] C. Côté, Y Brosseau, D. Létourneau, C. Raievsky, and F. Michaud. Robotic software integration using MARIE. *International Journal on Advanced Robotics Systems*, 3(1):55 – 60, 2006.

[28] C. Côté, D. Létourneau, F. Michaud, J. Valin, Y. Brosseau, C. Raievsky, M. Lemay, and V. Tran. Code reusability tools for programming mobile robots. *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2:1820 – 1825, 2004.

[29] B. D'Andréa-Novel, G. Bastin, and G. Campion. Modelling and control of non holonomic wheeled mobile robots. *Proceedings of the 1991 IEEE International Conference on Robotics and Automation.*, pages 1130–1135, 1991.

[30] B. D'Andréa-Novel, G. Bastin, and G. Campion. Dynamic feedback linearization of nonholonomic wheeled mobile robots. *Proceedings of the 1992 IEEE International Conference on Robotics and Automation*, pages 2527–2532, 1992.

[31] J.LL. de la Rosa, R. García, B. Innocenti, I. Muñoz, A. Figueras, J.A. Ramon, and M. Montaner. Rogi team real: Dynamical physical agents. *RoboCup-99: Robot Soccer World Cup III. Lecture Notes in Artificial Intelligence. Veloso, Pagello, Kitano (eds)*, 1856:434 –438, 2000.

[32] J.Ll. de la Rosa, B. Innocenti, M. Montaner, A. Figueras, I. Muñoz, and J.A. Ramon. RoGi team description. *RoboCup 2000: Robot Soccer. World Cup IV. Lecture Notes in Computer Science*, 2019:551 – 554, 2001.

[33] T. De Wolf. Panel discussion on engineering self-organising emergence. http://www.cs.kuleuven.be/∼tomdw/presentations/ presentation-SASOpanel2007.ppt, 2007. SASO 2007 10-07-2007, MIT, Boston/Cambridge, MA, USA.

[34] T. De Wolf and T. Holvoet. Using UML 2 activity diagrams to design information flows and feedback-loops in self-organising emergent systems. *Proceedings of the Second International Workshop on Engineering Emergence in Decentralised Autonomic Systems (EEDAS 2007)*, pages 52 – 61, 2007.

[35] S. A. DeLoach. Analysis and design using MaSE and agentTool. *Proceedings of the 12th Midwest Artificial Intelligence and Cognitive Science Conference (MAICS 2001)*, pages 1 – 7, 2001.

[36] M. Dorigo, V. Trianni, E. Sahin, R. Grob, T.H. Labella, G. Baldassarre, S. Nolfi, J.L. Debeubourg, F. Mondada, D. Floreano, and L. Gambardella. Evolving self-organizing behaviors for a swarm-bot. *Autonomous Robots*, 17:223 – 245, 2004.

[37] D. Driankov, H. Hellendoorn, and M. Reinfrank. *An Introduction to Fuzzy Control*. Springer-Verlag New York, Inc., 1993. ISBN:0-387-56362-8.

[38] FIPA. Fipa: Foundation for intelligent physical agents. on-line: http://www.fipa.org/, 2002.

[39] R.J. Firby. *Adaptive Execution in Complex Dynamic Domains*. PhD thesis, Yale University, 1989.

[40] K. Fischer, M. Schillo, and J. Siekmann. Holonic multiagent systems: A foundation for the organisation of multiagent systems. *Holonic and Multi-Agent Systems for Manufacturing. Lecture Notes in Computer Science.*, 2744/2004:71 – 80, 2004. ISBN 978-3-540-40751-5.

[41] J. Fromm. On engineering and emergence. *SAKS/06, Workshop on Adaptation and Self-Organizing Systems (nlin.AO)*, arXiv:nlin/0601002v1 [nlin.AO], 2006.

[42] T. Fukuda and Y. Kawauchi. Cellular robotic system (CEBOT) as one of the realization of self-organizing intelligent universal manipulator. *Proceedings of the IEEE International Conference on Robotics and Automation*, 1:662 – 667, 1990.

[43] R. Garcia, X. Cufí, and M. Carreras. Estimating the motion of an underwater robot from a monocular image sequence. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 3:1682 – 1687, 2001.

[44] E. Gat. Integrating planning and reaction in a heterogeneous asynchronous architecture for controlling mobile robots. *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI)*, pages 809–815, 1992.

[45] M.P. Georgeff and A.L. Lansky. Procedural knowledge. *Proceedings of the IEEE*, 74(10):1383 – 1398, 1986.

[46] B. Gerkey, A. Howard, and R. Vaughan. Player/Stage, 2005. http://playerstage.sourceforge.net/.

[47] B. Gerkey, R. Vaughan, and A. Howard. The Player/Stage project: Tools for multi-robot and distributed sensor systems. *Proceedings of the 11th International Conference on Advanced Robotics. Coimbra, Portugal,*, pages 317 – 323, 2003.

[48] B. Gerkey, R. Vaughan, K. Stoy, A. Howard, G. Sukhatme, and M. Mataric. Most valuable player: A robot device server for distributed control. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1226 – 1231, 2001.

[49] B.P. Gerkey, M.J. Mataric, and G.S. Sukhatme. Exploiting physical dynamics for concurrent control of a mobile robot. *Proceedings ICRA '02. IEEE International Conference on Robotics and Automation*, 4:3467 – 3472, 2002.

[50] A. Giret and V. Botti. Towards an abstract recursive agent. *Integrated Computer-Aided Engineering*, 11(2):165–177, 2004.

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

[52] I.J. Ha, M.S. Ko, and S.K. Kwon. An efficient estimation algorithm for model parameters of robotic manipulators. *IEEE Transactions on Robotics and Automation.*, Vol. 5, N3.:386–394, 1989.

[53] H.C.H. Hsu and A. Liu. A flexible architecture for navigation control of a mobile robot. *IEEE Transactions on Systems, Man, and Cybernetics: Part A: Systems and Humans*, 37(3):310–318, 2007.

[54] H. Hu and D. Gu. A multi-agent system for cooperative quadruped walking robots. *Proceedings of the IASTED International Conference Robotics and Applications.*, pages 1 – 5, 2000.

[55] B. Innocenti, B. López, and J. Salvi. A multi-agent architecture with cooperative fuzzy control for a mobile robot. *Robotics and Autonomous Systems*, (55):881 – 891, 2007.

[56] N. R. Jennings, K. Sycara, and M. Wooldridge. A roadmap of agent research and development. *Autonomous Agents and Multi-Agents Systems. Editorial Kluwer Academic Publishers*, pages 7–38, 1998.

[57] J. Jia, W. Chen, and Y. Xi. Design and implementation of an open autonomous mobile robot system. *Proceedings of the 2004 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1726 – 1731, 2004.

[58] V. Julian, C. Carrascosa, M. Rebollo, J. Soler, and V. Botti. SIMBA: An approach for real-time multi-agent systems. *Topics in Artificial Intelligence: 5th Catalonian Conference on AI, CCIA 2002, Castellon, Spain, October 24-25, 2002. Proceedings. Lecture Notes in Computer Science*, 2504/2002:283–292, 2002.

[59] L.P. Kaelbling, M.L. Littman, and M.L. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence Journal*, 101:99 – 134, 1998.

[60] G. A. Kaminka. Robots are agents, too! In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems*, 2007. Invited talk.

[61] G. A. Kaminka. Robots are agents, too! *AgentLink News*, pages 16 – 17, December 2004.

[62] B. Khoshnevis and G. A Bekey. Centralized sensing and control of multiple mobile robots. *Computers & Industrial Engineering*, 35(3-4):503 – 506, 1998.

[63] G. Kim and W. Chung. Tripodal schematic control architecture for integration of multi-functional indoor service robots. *IEEE Transactions on Industrial Electronics*, 53(5):1723 – 1736, 2006.

[64] M. Kolp, P. Giorgini, and J. Mylopoulos. Multi-agent architectures as organizational structures. *Autonomous Agents and Multi-Agent Systems*, 13:1–2, 2006.

[65] K. Konolige and K. Myers. *Artificial Intelligence and Mobile Robots: Case Studies of Successful Robot Systems*, chapter The Saphira Architecture for Autonomous Mobile Robots, pages 211 – 242. MIT Press Cambridge, MA, USA, 1998. ISBN:0-262-61137-6.

[66] K. Konolige, K. L. Myers, E. H. Ruspini, and A. Saffiotti. The saphira architecture: A design for autonomy. *Journal of experimental & theoretical artificial intelligence: JETAI*, 9(1):215–235, 1997.

[67] J. Kotzian and V. Srovnal. Design and optimization of distributed control system using UML model. *Proceedings of the 11th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems - ECBS'04*, pages 469 – 476, 2004.

[68] J. Kramer and M. Scheutz. Development environments for autonomous mobile robots: A survey. *Autonomous Robots*, 22:101 – 132, 2007.

[69] Mobile Robot Laboratory. Missionlab v6.0., 2003. http://www.cc.gatech.edu/aimosaic/robotlab/ research/MissionLab/.

[70] M. LaFary and C. Newton. Aria html documentation, 2005.

[71] P. Langley. Cognitive architectures and general intelligent systems. *AI Magazine*, 27:33 – 44, 2006.

[72] W. Leroquais. Modélisation et commande de robotsmobiles à roues en présence de pseudo-glissements". *Ph. D. Thesis, Êcole Nationale Supérieure des Mines de Paris*, 1998.

[73] Z. Li and C. H. Sim. A survey of emergent behavior and its impacts in agent-based systems. *IEEE International Conference on Industrial Informatics*, pages 1295 – 1300, 2006.

[74] M. Lindstrom, A. Orebäck, and H.I. Christensen. BERRA: A research architecture for service robots. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '00)*, 4:3278–3283, 2000.

[75] D. MacKenzie, R. Arkin, and J. Cameron. Multiagent mission specification and execution. *Autonomous Robots*, 4(1):29 – 52, 1997.

[76] V. Marik, M. Fletcher, and M. Pechoucek. Holons and agents: Recent developments and mutual impacts. *Multi-Agent-Systems and Applications II. 9th ECCAI-ACAI/EASSS 2001, AEMAS 2001, HoloMAS 2001. Lecture Notes in Computer Science*, 2322:233 – 267, 2003. ISBN: 978-3-540-43377-4.

[77] D.L. Martin, A.J. Cheyer, and D.B. Moran. Building distributed software systems with the open agent architecture. *Proc. of the Third International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM'98), London, UK*, pages 355–376, 1998.

[78] D.L. Martin, A.J. Cheyer, and D.B. Moran. The open agent architecture: A framework for building distributed software systems. *Applied Artificial Intelligence*, 13:91 – 128, 1999.

[79] M. Mataric. The basics of robot control. *online: http://www-robotics.usc.edu/∼maja/robot-control.html*.

[80] M. Mataric. Minimizing complexity in controlling a mobile robot population. *Proceedings of the 1992 IEEE International Conference on Robotics and Automation*, pages 830 – 835, 1992.

[81] M.J. Mataric. *Interaction and Intelligent Behaviour.* PhD dissertation. MIT., May 1994.

[82] A. Matsumoto, H. Asama, Y. Ishida, K. Ozaki, and I. Endo. Communication in the autonomous and decentralized robot system ACTRESS. *IEEE International Workshop on Intelligent Robots and Systems '90 (IROS'90). 'Towards a New Frontier of Applications'*, 2:835 – 840, 1990.

[83] Inc. MobileRobots. Activmedia robotics mobilerobots developer support, 2005. http://robots.mobilerobots.com/.

[84] F. Mondada, G. Pettinaro, A. Guignard, I. Kwee, D. Floreano, J. Debeubourg, S. Nolfi, L. Gambardella, and M. Dorigo. Swarm-bot: A new distributed robotic concept. *Autonomous Robots*, 17:193 – 221, 2004.

[85] M. Montemerlo, N. Roy, and S. Thrun. CARMEN, carnegie mellon robot navigation toolkit, 2003. http://carmen.sourceforge.net/.

[86] M. Montemerlo, N. Roy, and S. Thrun. Perspectives on standardization in mobile robot programming: The carnegie mellon navigation (CARMEN) toolkit. *IROS 2003. Las Vegas, NV*, 3:2436 – 2441, 2003.

[87] B. Moulin and B. Chaib-Draa. A review of distributed artificial intelligence. *Foundations of Distributed Artificial Intelligence, O'Hare, G. and Jennings, N. (eds), Wiley*, pages 3 – 55, 1996.

[88] J. Müller. A cooperation model for autonomous agents. In M. Wooldridge and N.R. Jennings, editors, *Intelligents Agents III (LNAI 1193)*, pages 245–260. Springer-Verlag, 1997.

[89] R.R. Murphy. *Introduction to AI Robotics*. The MIT Press, 2000.

[90] N. Muscettola, G. Dorais, C. Fry, R. Levinson, and C. Plaunt. IDEA: Planning at the core of autonomous reactive agents. Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space, October 2002, 2002.

[91] I.A.D. Nesnas, A. Wright, M. Bajracharya, R. Simmons, and T. Estlin. CLARAty and challenges of developing interoperable robotic software. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)*, 1:1/121–1/132, 2003.

[92] M. C. Neves and E. Oliveira. A multi-agent approach for a mobile robot control system. *Proceedings of Workshop on "Multi-Agent Systems:Theory and Applications" (MASTA'97 - EPPIA'97) - Coimbra -Portugal*, pages 1 – 14, 1997.

[93] Maria C. Neves and Eugénio Oliveira. A control architecture for an autonomous mobile robot. *Proceedings of First International Conference on "Autonomous Agents" (AA'97) , Marina del Rey, California, USA, Feb 1997*, pages 193 – 200, 1997. ISBN:0-89791-877-0.

[94] A. Oller. *Disseny D'agents Físics: Inclusió de Capacitats Específiques Per a L'avaluació de L'eficiència D'accions*. PhD thesis, Universitat de Girona, 2002. ISBN: 84-688-6471-4.

[95] A. Oller, J. Ll. de la Rosa, and E. Del Acebo. DPA: Arquitecture for co-operative dynamical physical agents. *MAMAAW'99*, June 1999.

[96] D. Omerdic and G. Roberts. Thruster fault diagnosis and accommodation for open-frame underwater vehicles. *Engineering Practice*, 12:1575–1598, 2004.

[97] A. Orebäck and H.I. Christensen. Evaluation of architectures for mobile robotics. *Autonomous Robots*, 14:33 – 49, 2003.

[98] OROCOS. on-line: http://www.orocos.org.

[99] R.T. Pack, D.M. Wilkes, and G. Kawamura. A software architecture for integrated service robot development. *IEEE International Conference on Systems, Man, and Cybernetics, 1997. 'Computational Cybernetics and Simulation'*, 4:3774 – 3779, 1997.

[100] L. E. Parker. L-ALLIANCE: Task-oriented multi-robot learning in behavior-based systems. *Advanced Robotics, Special Issue on Top Selected Papers from International Conference on Intelligent Robots and Systems*, 11(4):305 – 322, 1997.

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

[102] G.A.S. Pereira, M.F.M. Campos, and L.A. Aguirre. Data based dynamical modeling of vision observed small robots. *2000 IEEE International Conference on Systems, Man, and Cybernetics.*, vol.5:3312 – 3317, 2000.

[103] P. Pirjanian. Behavior coordination mechanisms – state-of-the-art. Technical Report IRIS-99-375, Institute of Robotics and Intelligent Systems, School of Engineering, University of Southern California., 1999.

[104] Paolo Pirjanian. An overview of system architecture for action selection in mobile robotics. *Tech-report, Laboratory of Image Analysis, Aalborg University*, 1997.

[105] R. Rajagolapan and N. Barakat. Comparative study of velocity and computed torque control schemes for a differentially driven automated vehicle. *Proceedings of the 1996 IEEE International Conference on robotics and Automation. Minessota, U.S.A.*, pages 3637–3643, 1996.

[106] P. Ridao, J. Batlle, and M. Carreras. O2CA2, a new object oriented control architecture for autonomy: the reactive layer. *Control Engineering Practice*, 10(8):857–873, August 2002.

[107] P. Ridao, A. Tiano, A. El-Fakdi, M. Carreras, and A. Zirilli. On the identification of non linear models of unmanned underwater vehicles. *Control engineering practice*, 12(12):1483 – 1499, 2004. ISSN 0967-0661.

[108] Planas R.M., Fuertes J.M., and Martínez A.B. Qualitative approach for mobile robot path planning based on potential field methods. *Sixteenth International Workshop on Qualitative Reasoning QR2002*, 2002. On-line: http://www.upc.edu/web/QR2002/Papers/QR2002

[109] ActivMedia Robotics. *Pioneer 2 / People Bot: Operations Manual*. ActivMedia Robotics, 2002.

[110] University of Ulm Robotics Group. Miro - middleware for robots, 2005. http://smart.informatik.uniulm. de/MIRO/index.html.

[111] J. K. Rosenblatt. *DAMN: A Distributed Architecture for Mobile Navigation*. PhD thesis, Robotics Institute at Carnegie Mellon University, 1997.

[112] R. J. Ross. Research proposal: Development of a MAS based robot control architecture and the investigation of speech priming on robots. *University College Dublin*, 2002.

[113] A. Saffiotti. The uses of fuzzy logic in autonomous robot navigation. *Soft Computing*, 1(4):180 – 197, 1997.

[114] M. Scheutz. ADE - steps towards a distributed development and runtime environment for complex robotic agent architectures. *Applied Artificial intelligence*, 20(4-5):275 – 304, 2006.

[115] A. Silva and J. Delgado. The agent pattern: A design pattern for dynamic and distributed applications. *Proceedings of the EuroPLoP'98, Third European Conference on Pattern Languages of Programming and Computing, Irsee, Germany*, 1998. Article ID: 2836264.

[116] R. Simmons. The curvature-velocity method for local obstacle avoidance. In *International Conference on Robotics and Automation*, April 1996.

[117] R.G. Simmons. Structured control for autonomous robots. *IEEE Transactions on Robotics and Automation*, 10(1):34–43, 1994.

[118] S. P. N. Singh and S. M. Thayer. ARMS: Autonomous robots for military systems. a survey of collaborative robotics core technologies and their military applications. Technical report, CMU-RI-TR-01-16, The Robotics Institute - Carnegie Mellon University, 2001.

[119] M.G. Slack. Navigation templates: Mediating qualitative guidance andquantitative control in mobile robots. *IEEE Transactions on Systems, Man and Cybernetics*, 23(2):452 – 466, 1993. ISSN: 0018-9472. Digital Object Identifier: 10.1109/21.229458.

[120] L-K. Soh and C. Tsatsoulis. A real-time negotiation model and a multi-agent sensor network implementation. *Autonomous Agents and Multi-Agent Systems*, pages 215–271, November, 2005.

[121] W. Spears, D. Spears, J. Hamann, and R. Heil. Distributed, physics-based control of swarms of vehicles. *Autonomous Robots 17*, pages 137 – 162, 2004.

[122] K.P. Sycara. Multiagent systems. *AI MAGAZINE - American Association for Artificial Intelligence*, pages 79–92, 1998.

[123] Y. Tahara, A. Ohsuga, and S. Honiden. Agent system development method based on agent patterns. *ICSE '99: Proceedings of the 21st international conference on Software engineering.*, pages 356 – 367, 1999. ISBN: 1-58113-074-0.

[124] B. Thuilot. Contribution à la modélisation et à la commande de robots mobiles à roues. *Ph. D. Thesis. Êcole Nationale Supérieure des Mines de Paris.*, 1995.

[125] A. Tiano, M. Carreras, P. Ridao, and A. Zirilli. On the identification of non linear models of unmanned underwater vehicles. *10th Mediterranean Conference on Control and Automation. Lisbon, Portugal*, 2002.

[126] M. Tounsi, M. Gautier, and W. Khalil. Identification of dynamic parameters of mobile robot. *ICAR95*, pages 333–340, 1995.

[127] M. Tounsi, G. Lebret, and M. Gautier. Dynamic control of a nonholonomic mobile robot in cartesian space. *Proceedings of the 34th Conference on Decision and Control*, pages 3825–3830, 1995.

[128] C. Tranoris and K. Thramboulidis. Integrating UML and the function block concept for the development of distributed control applications. *Proceedings of the IEEE Conference of Emerging Technologies and Factory Automation - ETFA '03*, 2:87 – 94, 2003.

[129] H. Utz, S. Sablatnög, S. Enderle, and G. Kraetzschmar. Miro: Middleware for mobile robot applications. *IEEE Transactions on Robotics and Automation, Special Issue on Object-Oriented Distributed Control Architectures*, 18(4):493 – 497, 2002.

[130] M. Veloso, J. Carbonell, A Pérez, D. Borrajo, E. Fink, and J. Blythe. Integrating planning and learning: The PRODIGY architecture. *Journal of Experimental and Theoretical Artificial Intelligence*, 7(1):81 – 120.

[131] G. Weiss. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. The MIT Press, Cambridge, Massachusetts, 1999.

[132] M. F. Wood and S. A. DeLoach. An overview of the multiagent systems engineering methodology. *Lecture Notes in Computer Science. Vol. 1957/2001, Springer Verlag.*, pages 207 – 221, 2000.

[133] M. Wooldridge. *An Introduction to Multiagent Systems*. John Wiley & Sons, LTD, 2002.

[134] H. Yavuz and A. Bradshaw. A new conceptual approach to the design of hybrid control architecture for autonomous mobile robots. *Journal of Intelligent and Robotic Systems 34: 1-26, Kluwer Academic Publishers.*, 2002.