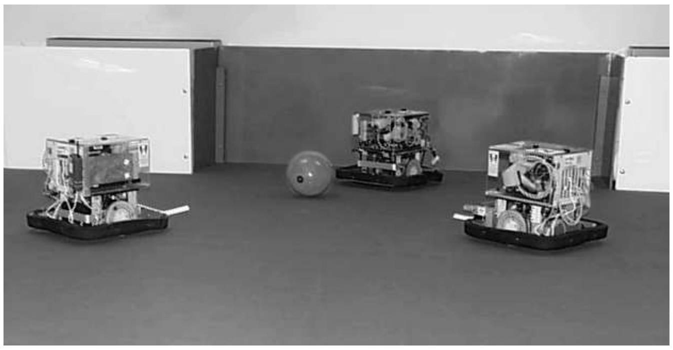
UNIVERSIDADE TÉCNICA DE LISBOA
INSTITUTO SUPERIOR TÉCNICO



DESIGN AND IMPLEMENTATION OF A POPULATION OF COOPERATIVE AUTONOMOUS ROBOTS

Pedro Miguel Martins da Silva Aparício
(Licenciado)

Dissertação para obtenção do Grau de Mestre em Engenharia Electrotécnica e de Computadores

Orientador Científico:
Doutor Pedro Manuel Urbano de Almeida Lima

Constituição do Júri:
Doutor Pedro Manuel Urbano de Almeida Lima
Doutora Maria Manuela Magalhães de Albuquerque Veloso
Doutor Carlos Manuel Ribeiro Almeida
Doutor João Fernando Cardoso Silva Sequeira

Lisboa, Janeiro de 2000

À Elisabete,

RESUMO

Esta tese descreve o projecto e implementação de uma população de robots móveis autónomos, destinados a executar tarefas que exijam cooperação. A arquitectura funcional usada desenvolve-se em três níveis funcionais: Organizacional, Relacional e Individual. O nível Organizational encarrega-se de estabelecer a *estratégia* a seguir pela equipa, dada uma configuração do estado do mundo, de modo a atingir um objectivo. Os agentes negociam entre si a nível Relacional de forma a chegar a acordos. A nível Relacional é ainda feita a tradução entre a *estratégia* prescrita a nível organizacional e a *táctica* a implementar. A nível Individual, o agente assume um determinado *papel* (comportamento mais posição no campo). Todo o controlo de baixo nível é realizado neste nível.

No âmbito deste trabalho foi desenvolvida uma infra-estrutura composta por robots móveis autónomos com capacidades de processamento, sensoriais e de actuação que permitem a sua utilização como plataformas de investigação científica. Estas plataformas foram desenvolvida de raíz, tendo-se usado o maior numero possível de componentes de electrónica de consumo.

São apresentados e discutidos resultados obtidos com a população de robots no domínio do futebol robótico.

Palavras Chave: Robótica Cooperativa, Arquitecturas de Controlo Inteligente, Robótica Móvel, Sistemas Multi-Agente, RoboCup.

ABSTRACT

This thesis addresses the design and implementation of a population of cooperative autonomous robots. The functional architecture has three levels: Organizational, Relational and Individual. The Organizational level establishes the *strategy* to be followed by the team, given the world state, in order to accomplish an objective. At the Relational level, groups of agents negotiate and eventually come to an agreement on the *roles* to exhibit. Based on the strategy prescribed by the Organizational level, *roles* are distributed by the different agents, according to some *tactics*. The Individual level encompasses roles implementation (*behaviors plus actuation zones*) and handles all low-level control issues.

The developed infrastructure includes a population of mobile, fully autonomous robots, with on-board processing, sensing, actuation and comunications capabilities. The robots were developed to be used in research projects. All development was done from scratch, and consumer electronics components were used whenever possible.

Results obtained with the population of soccer robots are presented and discussed in the thesis.

Key Words: Cooperative Robots, Intelligent Control Architectures, Mobile Robots, Multi-Agent Systems, RoboCup.

Acknowledgments

In first place I want to thank my advisor and friend, Prof. Pedro Lima for his constant support and motivation. Without him this work would not have been done.

Rodrigo Ventura was also very important, with him I have spent great times of learning, discussing and hard working. He is responsible for the coding of many of the ideas expressed in this thesis.

To my family, I thank for their patience for my un-attendances and support during this work.

Professors Maria Isabel Ribeiro, Carlos Pinto-Ferreira and Luís Custódio and my friends at the Intelligent Control Laboratory at *Instituto de Sistemas e Robótica* have been very supportive.

I am thankful to the *Secção de Sistemas e Controlo do Departamento de Electrotecnia do Instituto Superior Técnico* for their support with ideas and facilities that have been very important for this work. The *Instituto de Sistemas e Robótica* provided the infrastructure I have used and the good environment it provides to its researchers was valuable.

The Portuguese Foundation for Science and Technology has supported this work throught the scholarship BM/12937/97.

Contents

List of Figures

Chapter 1

INTRODUCTION

Mobile Robots can be found nowadays in factories, storage areas, universities, hospitals, nuclear plants, private homes and even on *Mars*. Multi-robot populations [7] [17] [59] are becoming increasingly popular among researchers. Many competitions are held regularly, where populations of mobile robots work towards the accomplishment of a given (common) task (e.g., AAAI Robotic Contest [8], RoboCup [53] and MiroSot [45]). From a commercial/industrial standpoint, this *distributed* implementation of a robot is an interesting solution. Several robots working together, to move heavy and hazardous loads [1], to deliver mail and small parcels in office environments [21], to build maps of unknown environments [51], to detect landmines [23] or to transport food and medicines inside hospitals [56], are an advantageous solution with respect to a one-robot system. Advantages include: robustness; scalability; broad actuation area; parallel execution and problem solving and simplicity.

In multi-agent robotic research the challenges are not small or easy to overcome. The design of these systems involves knowledge from many different areas, such as Artificial Intelligence, Control, Electronics, Mechanics and others. Often, the application domain is not structured and is plenty of perturbations, such as static and mobile obstacles (e.g., other robots). When many robots join together in a confined space, one of the biggest challenges is the control of the individuals in order to avoid interference and resource sharing problems. Another major challenge is the maintenance of a robot population working in good conditions, i.e., reliably. A single robot is hard to maintain. Many robots are even harder.

This thesis addresses the problems related to the design and implementation of a population of cooperative autonomous robots.

The work was carried out within the framework of a joint project between the Intelligent Control Laboratory and the Artificial Intelligence and Manufacturing Systems Laboratory at the Institute for Systems and Robotics (ISR) of Instituto Superior Técnico (IST).

1.1 Motivation

This work aims at capitalizing past experience in the design of mobile robots [4], agent programming [72] and on the design of industrial prototypes [31] [52]. Past experience was focused on single robot systems. Nevertheless, that experience was very important for the design and construction of multi-agent robotic systems, capable of executing cooperative tasks.

At this point, a note must be made on the exact type of multi-agent systems that is discussed in this work. Multi-Agent Robotic Systems (MARS) are different from other multi-agent systems (MAS) because they have to interact with the real world, which is difficult to model. According to Fukuda [25], the behavior of a robotic agent depends on its perceptions and on the physical constraints from the environment and other agents. He concludes that the issue of cooperation involves physical contacts, time and space restrictions.

The main motivation to design this robotic population was its application to robotic soccer, namely to participate in Robocup. RoboCup is the Robot Soccer World Cup. Organized on an annual basis by the RoboCup International Federation, it is a contest that aims at fostering intelligent robotics research by providing a standard problem where a wide range of technologies can be integrated and examined. Robocup is a good example of how fast a good idea progresses. In the first edition, in 1997, 6 teams were present at the middle size contest. In 1998, 16 teams applied for the competition. In 1999, more that 24 have applied. Such an interest is revealing. It is motivated by the fact that it provides researchers with a common testbed, where different solutions may be tested and compared. Another advantage of such a competition is the necessity to build real hardware and software that works in an adverse world.

1.2 Objectives

The following is a summary of the main objectives of the work addressed in this thesis:

- To *build and control* a population of autonomous, cooperative, mobile robots.
- To *show the results* in a particular case study – *RoboCup*.
- To provide the developed population with enough *flexibility* to cope with other application domains, such as garbage collection, mail delivery and food foraging.

- To foster public interest in science and technology by demonstrating the robotic population in international scientific events.

1.3 Original Contributions

A re-interpretation of the *Behavior* concept is introduced. In this work a Behavior is presented as set of purposive states (i.e., with a goal) forming a finite state machine where *state* is proposed to be redefined as a STA loop, i.e., a sense-think-act loop that embodies *a goal, a sensing mechanism, an algorithm to be used and a actuation function.* The concept has been used in the implementation of the control architecture.

1.4 Thesis Outline

Chapter 2 presents some of the contributions related to Multi-Agent Robotic Systems found in the literature. In Chapter 3 the functional architecture designed for the control of the population is introduced. Chapter 4 presents a case study for the application of this architecture – Soccer Robots. Implementation issues are discussed in this chapter. In Chapter 5 the developed testbed (including the robots) is presented. The results obtained are presented and discussed in the same chapter. Conclusions are drawn in Chapter 6, together with plans for future work. Appendix A details the robots hardware and Appendix B presents some notes on the software developed.

Chapter 2

MULTI-AGENT ROBOTIC SYSTEMS

Building Multi-Agent Robotic Systems (MARS) is a challenging task due to the many options open to the designer, regarding almost every issue involved, from the kinematic structure to the control architecture.

Only recently, the robotics and artificial intelligence research communities have focused on the synthesis and control of those systems as a new research area, which gathers knowledge from many fields (e.g., Distributed Systems, Computer Science, Artificial Intelligence, Robotics and Control).

When compared with single-agent systems, MARS offer a number of advantages. According to Stone and Veloso [60] and Arkin [6], those include:

- *Robustness* – Being distributed and parallel systems by nature, MARS are less prone to failures. Even if an agent is damaged, the overall performance will not be compromised (provided that there is redundancy).

- *Scalability* – MARS have the potential to be *open* systems. One may introduce extra agents without major changes to the overall system;

- *Broad geographic area* – The agents can be separated over a wide geographic area, i.e., MARS can accomplish tasks that a single robot could not.

- *Divide and Conquer* – Many problems are well suited to be solved by several mobile robots.

- *Simplicity* – MARS are modular, making the programming and testing of the individual sub-systems easier.

- *Testbed* – MARS may be used as a testbed for the study of areas such as social sciences and Distributed Artificial Intelligence.

Despite the advantages, MARS have some drawbacks:

- Coordination between the agents is difficult. Interferences may exist.
- The performance of a robotic team is not easy to measure and therefore improvements are difficult to measure.
- Robot hardware is unreliable and, as a result, it is very hard to maintain a team of *operational* robots.

Different approaches have been developed to the synthesis and control of Multi-Agent Robotic Systems. Among them, the major features range from the group architecture (comprising the decision making mechanism and the communication scheme), to the ability to model other agents and objects in the environment. The use of machine learning to better adapt the population actions to the working environment also distinguishes some of the approaches.

In the sequel, some of the work available in the available literature is classified according to those features and briefly described.

2.1 Group Architecture

The *group architecture* is the infra-structure underlying the population organization. It comprehends concepts such as homogeneity of the population, the population control architecture and the communication capabilities [17]. Ultimately, the architecture determines the abilities and limitations of the system [39]. In the following, some of the key features of the group architecture are presented.

2.1.1 Decision Making

One of the fundamental decisions to be taken when defining the group architecture is the decision making structure. At design time, the options for the decision making [34] system are:

- *Centralized* decision – Given a task, decisions are made in a central agent and sub-tasks are transmitted to the executive units. This requires that the decision agent has access to all possible information.
- *Distributed* decisions – Given a task to the system, each agent makes its own decisions and executes them autonomously (eventually with help from the others agents).

- *Decentralized* decisions – Decisions are the result of a negotiation process among the system agents and are executed by them. No intervention of a centralized system exists.

In MARS, centralized decision systems are often associated with plan based systems. Russell and Norvig [55] presents examples of such systems. Given a goal and the population state, the central system plans the actions for the robots composing the population. One of the major problems with this approach is the potential to have dead-lock or bandwidth problems.

Noreils and Chatila [46] argue that if the environment changes too rapidly with respect to planning time, then the need for planning may be questionable. Usually, planning requires the world be predictable to some extent. In most situations this is the case, i.e., the world changes at a slow pace. If that is not the case, the system must be reactive to some extent in order to react to relevant events.

In [2], Alamis *et al.* present a cooperation system where robots incrementally merge their plans. The exchanged information regards the robots current state and future actions. It is a centralized system, well suited to the control of large number of robots navigating is a well-defined route network. Results on the control of real robots are presented in [1].

Distributed control systems are often associated with behavior-based and reactive solutions, inspired by Brooks' work [15].

In and attempt to attain a compromise between purely reactive and deliberative approaches, *hybrid controllers* have emerged as an alternative for the control of MARS. Those systems typically use a reactive system for low-level control (i.e., to handle unexpected events) and a planner for higher-level decision making (i.e., to deliberate the goals to achieve).

Matarić [40] presents an implementation of such an architecture for the control of a mobile robot. The robot objective was to explore an office environment while constructing and maintaining a map based on landmark discoveries. Yokota *et al.* [73] presents a decentralized architecture for the control of a population of mobile robots, where communications have a major role.

Distributed and decentralized control architectures are also common in the area of computer architectures. Lueth and Laengle [34] and Matarić [42] claim that these architectures have inherent advantages over the centralized architectures, namely:

- *Modularity* – For a well defined framework, it is possible to develop and test single parts of the system independently (hardware or software). This also eases maintenance when there is a malfunction in a subsystem.

- *Decentralized Knowledge* – Each subsystem handles its local knowledge, keeping only the relevant information. This has the advantage that information is shared only when required, thus reducing bandwidth.
- *Fault-tolerance and Redundancy* – Components of non-centralized systems are not likely to fail simultaneously. Robustness can be increased by adding redundancy to the system (e.g., a system with two mobile robots with the same capabilities may allow the system to continue working even if one of the robots fails).
- *Scalability* – Integration of new modules or agents, with new functionalities, does not imply a change in the control architecture (e.g., the introduction of a robot equipped with a manipulator arm in a population of robots).
- *Exploitation of Parallelism* – All agents work simultaneously. In many cases, problem solving is faster.

2.1.2 Homogeneity vs Heterogeneity

Homogeneity concerns the capacities of the robots that belong to the population. Should all robots be equal, the population is said to be *homogeneous*. If that is not the case, the population is *heterogeneous* (e.g., a population where different robots have different actuators). According to Cao *et al.* [17], heterogeneity adds complexity to the system, leading to increased difficulties in task allocation.

Heterogeneous systems inherently lead to different agents having different roles in the system. An example of such a system is presented by Birk and Belpaeme in [14]. In homogeneous systems, the sole differences are on the agents roles (which may change dynamically).

2.1.3 Communications

The communication scheme is decisive to the population performance. The most common segmentation, presented in Cao *et al.* [17], classifies the communication schemes as follows:

- *via Environment* – This communication scheme uses the environment as a *blackboard* where the different agents interact, using common, a-priory defined signals. This is a limited communication scheme as the group of possible signals is limited and the communication does not have an explicit destination (may not be received at all). An example is a box-moving application. Imagine that robot R has the task of moving boxes from place B1 to B2. It may use the absence of a box in B2 as a signal to indicate that it is necessary to pick a box in B1 and move it to B2. Fujii *et al.* [24] present results of such a system where communication is done through the environment;

- *via Sensors* – This is an inherently local communication scheme. Agents rely on the sensors to communicate. Modeling of the environment and other agents is mandatory to use this communication scheme. Common applications are pattern formation and robot-following applications. Imagine a robot RA that is instructed to follow another robot, RB. RA uses its sensorial information to 'see' if RB is in front of it and if not, tries to locate it and move next to it. Populations of robots with such instructions can be used to move heavy equipment in industrial environments. Note that RA would need the capability to model other robots (namely RB);

- *direct Communications* – Direct communications is the easiest way to pass messages between robots. These can be implemented by either direct or broadcast mode. Architectures based on this communications scheme are similar to computer networks and many issues regarding protocols and topologies use computer network concepts. An example robot population using such a communication scheme is presented in [65].

Other authors, such as Arkin [6] classify the possible communication schemes in *explicit* and *implicit*. Those can be related to the previously presented ones.

Communication via environment and via sensors is susceptible to errors generated by misunderstanding and misinterpretation of the situation as shown by Yokota *et al.* in [73]. They are, therefore, of limited use. Nevertheless, in structured environments where unexpected events are rare and the necessary bandwidth is small, this may be an acceptable solution.

In [38], Matarić proposes a methodology to use communications to share data between the agents, thus reducing the locality effect inherent to geographically sparse MARS. Ohko *et al.* [47] describe a method that reduces the communications bandwidth needs within a team of mobile robots by using case-based reasoning.

Premvuti and Yuta [50] proposes a division of cooperation of robots in two types: *active cooperation* and *non-active cooperation*. Active cooperation exists if all the robots in the system take an action in order to do something that creates a progressive result, such as increasing performance. On the contrary, an attitude that is not of such nature is called *non-active cooperation*.

2.1.4 Other Classification Lines

Other authors, such as Dudeck *et al.* [26] propose a taxonomy to classify MARS according to the following lines:

- *Team size* – Refers to the number of autonomous agents in the collective.
- *Communication range* – Translates the robots ability to communicate with each other. Comprises the *none*, *near* and *infinite* subclasses.
- *Communication topology*– The allowed communication paths. Include broadcast, point-to-point, tree and graph configurations.
- *Communication bandwidth* – Refers to the limit on the amount of information that may be exchanged. Ranges from zero to infinity.
- *Team reconfigurability* – Deals with the flexibility regarding the structure and organization of the population.
- *Processing ability* – Refers to the underlying computational model (e.g., finite state machine).
- *Team Composition* – Heterogeneous or homogeneous robots.

According to Balch [7] this taxonomy is becoming an important reference for multi-agent systems classification because it provides a useful set of axes for the classification of multi-agent robot systems.

2.2 Modeling

Modeling other agents deals with understanding and representing the other agents intentions, states and actions.

Matarić [41] showed that the ability to distinguish other robots from the rest of the objects allows the overcoming of interference through the use of simple avoidance mechanisms.

The issue is very important in those systems where direct communication does not exist. If direct communication exists, modeling can be used to reduce the communication load. On the other side, modeling other agents requires more computational power (which is limited in most cases).

Communications can be used to help modeling other agents because situated agents cannot sense all the information required to model other agents states. Hidden state problems can thus be overcome. This is useful in applications where competing agents are present. If one of the robots in a team detects an undesirable agent and communicates that information to other team members, those will have a better world model.

Even if a robot does not model other robots, modeling always exists to identify obstacles, objects or goals.

Teamwork models have been proposed [18] to provide a model of explicit cooperative problem solving that specifies how groups of agents should operate and interact in complex, dynamic environments in order to achieve a goal. It solves the problem outlined by Lueth and Laengle in [34] that regards the fact that pure distributed and decentralized decision making structures systems have difficulties to guarantee that the overall system goals have been achieved.

The joint intentions framework has been proposed by Cohen et Levesque in [18] to model teamwork. This framework has been further developed, implemented and tested by other researchers such as Tambe [61] [62] and Jennings [27].

Such models for teamwork solve problems such as conflicts and interferences among team members. Conflicts exists whenever several robots compete for a unique resource. Examples of conflicting resources are: a single battery charger, maneuver space or critical information and communication bandwidth. In an implementation where *teamwork* models were not used, Schneider-Fountán and Matarić [58] showed that the performance of a group of mobile robots foraging food reduces with a increase in the number of agents interacting, mainly due to interferences among the robots.

2.3 Learning

Learning is used when the designer has incomplete knowledge of the environment and the determination of control parameters is difficult, time consuming or impossible for a human operator to do [55]. Other authors argue that learning can improve performance in many ways because it introduces new knowledge, allows for the generalization of new knowledge and reorganizes system information (Arkin [6]).

In MARS, learning can be used at different levels, from the lower level (single robot control parameters), to the high level control parameters (population control parameters). Stone [59] and Stone and Veloso [60] addressed this question and presents a study on the application of multi-layer machine learning within multi-agent systems, namely, the robotic soccer domain. Other approaches to learning in MARS include Balch work [7].

Up to this point, the reinforcement learning machine learning technique has been the most popular technique. Examples of its application to MARS include

Matarić [42], [37], Parker [49] and Asada *et al.* [63]. In [43] Matarić describes how, given a set of basic behaviors, social behaviors can be learned using a combination operators. Learning applied to Task-Oriented MARS is presented in [49].

Uchibe *et al.* [64] present results on the application of reinforcement learning to the soccer domain. In this application, a player learned to shoot a ball at a goal.

In [36] Dillmann *et al.* propose the use of communication to learn in MAS. It assumes that there is no a-priori knowledge of a formalism for individual knowledge, but a common agent-independent language. In this work, agents further share a common ontology, i.e., when an agent receives a message, it understands exactly what it means.

Techniques inspired in nature (Genetic Algorithms) have also been used, mostly in simulation work. Luke and Spector [35] present an application where genetic models were used to control a team of simulated robots in a *predator/pray* domain.

Chapter 3

FUNCTIONAL ARCHITECTURE

This chapter presents the reference functional architecture designed to control a population of mobile robots. The functional architecture is overviewed first. To illustrate some of the concepts, examples from the soccer domain are introduced. The key definitions of this work are stated in this chapter. The details are presented in the sequel.

3.1 Team Functional Architecture

The population control architecture defines the way the robot population, as a whole, represents goals, senses, processes information and acts [48]. It has a strong effect on the system performance and is used as an abstraction to handle complexity.

The architecture is inspired on a 3-level functional hierarchy, proposed by Drogoul and Collinot [20] to model a team of autonomous agents. This architecture comprises three levels of competence. Our particularization of that control architecture for the control of a multi-agent robotic system follows (see Figure 3.1 for a schematic view):

- **Organizational**: This level establishes the *strategy* to be followed by the team, given the world state (team plus mission status) to attain an objective.

- **Relational**: At this level, groups of agents negotiate and eventually come to an agreement. Based on the strategy prescribed by the organizational level, *roles* are distributed by the different agents, according to some *tactics*. Agreements are based on robot abilities, state and the result of negotiation.

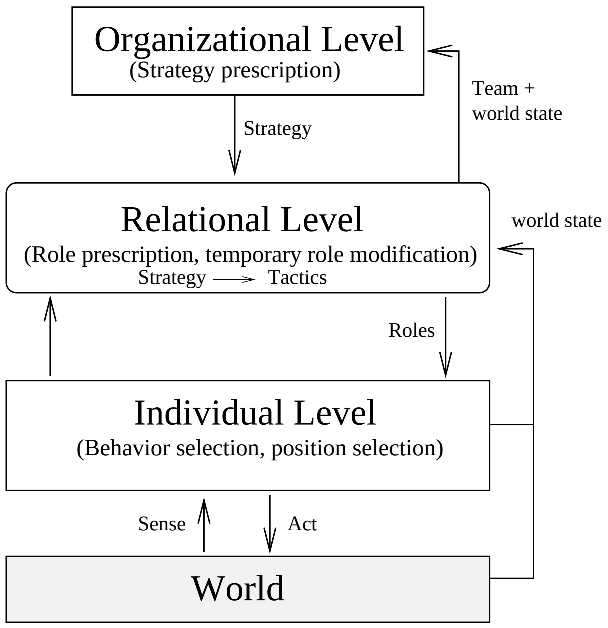


Figure 3.1: Team Functional architecture

- **Individual**: This level encompasses all the available roles (*behaviors plus actuation zones*). The behaviors are composed of the *primitive tasks* (e.g., kick, follow object, path planning, motion with collision avoidance) and their relations. All low-level control issues are handled at this level.

In the *Management* domain [9] it is common to classifie the nature of organizations in 3 levels: *top management* (organizational perspective); *middle management* (department perspective); *first-level management* (individual perspective). Those levels are associated with the following goals:

- *strategic goals* are targets set by top management. Usually address issues relating to the organization as a whole and many times are stated in fairly general terms;

- *tactical goals* are set by middle-management for specific department units. Goals at this level spell out what must be done by the various departments in order to achieve the the results outlined by the strategic objectives;

- *operational goals* are set by lower management levels. Address specific, measurable outcomes required from lower levels.

Control researchers have used this approach in the past to model the control of complex systems, referred as Intelligent Control [57].

3.1.1 Organizational Level

The organizational level is responsible for the overall control of the population. In a sense, it imposes a set of rules and constraints that enable the robots to act as a team, in order to fulfill the mission. It decides on the *strategy* to be followed according to a given *world state* and application (the possible strategies and their relations are defined by a human operator/programmer). In this context,

Definition 1 Strategy *is the action to take (*`what to do`*) for interacting with the environment in order to achieve a long term goal.*

For instance, in the soccer domain, the environment is composed of the world state (information regarding the players and their environment) and the game state. In that domain the strategy comprises concepts such as `defend`, `counter-attack` and `attack`. In a domain such as formation control, the strategy could comprise concepts such as `diamond`, `wedge`, `line` or `column`. Note that the different strategies are abstract concepts, i.e., no prescription on the control of an agent is done at this level. Only the decision on *what to do* is transmitted.

Although the organizational level is a central decision system, it differs from the common ones in the sense that the centralizing agent may be replaced by another (dormant in normal operation) if an exception arises. This level is implemented in all robots of the population, but only one of the robots has this level active at a time. Tambe [62] showed that, in some applications, an agent that has an overview of the team is necessary. This is the robot where the organizational level is active.

3.1.2 Relational Level

The relational level is responsible for the translation of the orders sent from the organizational level (the *strategies* to be followed) into executable actions (*tactics*), i.e., the relational level translates `what to do` prescribed at organizational level into `how to do` orders. At this level, *tactics* is defined as:

Definition 2 Tactics *represents a mapping between strategies and pre-determined team configurations composed of roles, where* role *is defined as:*

Definition 3 Roles *are* position/behavior *pairs to be undertaken by individual robots.*

This level runs simultaneously in all team members, and it is the level at which the agents are aware of each other and interact.

Given the strategy to be followed, the active agents implement the prescribed *tactics*, using negotiation at the relational level, to decide who should move where and what role to exhibit. Conflicts between agents are detected and solved at this level.

Besides the translations, all negotiations between agents go through the relational level in order to keep the group of agents as a team. For instance, in the soccer domain, if two forward players are trying to get the ball simultaneously, they should come to an agreement, at relational level, determining who should go for the ball in the first place (e.g., the one closest to the ball). These agreements have local scope (only the agents involved are aware of it).

3.1.3 Individual Level

In each agent, the individual level implements a *role* negotiated at the relational level. In a sense, it maps the relational level `how to do` into `doing`. Control issues related to sensor readings, data processing and actuation are handled at this level. Issues related to collision avoidance and other reactive actions are also handled at this level.

Roles implementation is done at the individual level. Possible roles within the soccer domain include Goal-Keeper, Defender, MidField and Forward. In the formation control domain, possible roles would be *Formation_Leader*, *Formation_Agent*, *Formation_Tail*.

3.2 Implementation Issues

This section presents details on the implementation of the functional levels described in the previous section. The architecture has not been fully implemented yet.

3.2.1 Organizational Level

This level embodies the team *long term* goals and the way to accomplish them. It is implemented as a *state machine* whose states correspond the *strategy* applicable to a given situation. The translations between *strategies/states* are traversed upon changes on the world state, which lead to different responses. Figure 3.2 presents a schematic view of an organizational level state machine implementation.

The world state upon which conditions are evaluated is stored in the common *blackboard*. It results from the post-processing of raw data, acquired at lower levels (Individual and Relational).

According to the changes in the world state, and to the current strategy, a given strategy is prescribed to the team, corresponding to the recipe that should be applied to that situation. For example, in the soccer domain, if our *current*

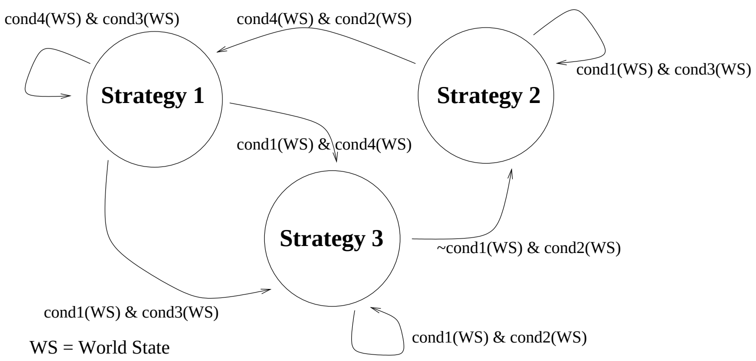


Figure 3.2: Example of an Organizational Level State Machine.

strategy (state) is `defense` and the world state variables *ball_is_in_our_field* and *one_of_our_players_controls_the_ball* are true, a possible strategy to follow would be a `counter-attack`. That strategy would be prescribed to the relational level upon the transition from the state `defense` to `counter-attack`, enabled by the `TRUE` evaluation of the conditions in that transition. This prescription should, hopefully, lead the team to move into the opponents field and try to score a goal.

The Organizational Level operation is a mapping between a set of *states* (current strategy), conditioned by a set of logical conditions on the world state. The 4-tuple (Q, S_c, L, δ_s), represents these relations, where the symbols have the following meanings:

Q : set of *states*, each corresponding to a *strategy*.

S_c : the initial state or strategy.

L : set of logical conditions applicable to the current world state

δ_s : the *next_strategy* function

The δ_s function is defined over the cartesian product of the set of logical conditions and the set of strategies. The result is the prescribed strategy.

$$\delta_s : Q \times L \rightarrow Q$$

3.2.2 Relational Level

Figure 3.3 presents a schematic view of the relational level implementation. As stated before, when a strategy is received, it is converted into some (pre-defined) tactics that was a priory defined by the team manager. The tactics is stored at the relational level, and it is common to all the agents.

An example mapping for the soccer domain (4 robot teams) is presented in Table 3.1. It shows the roles that the robots have to exhibit in order to implement

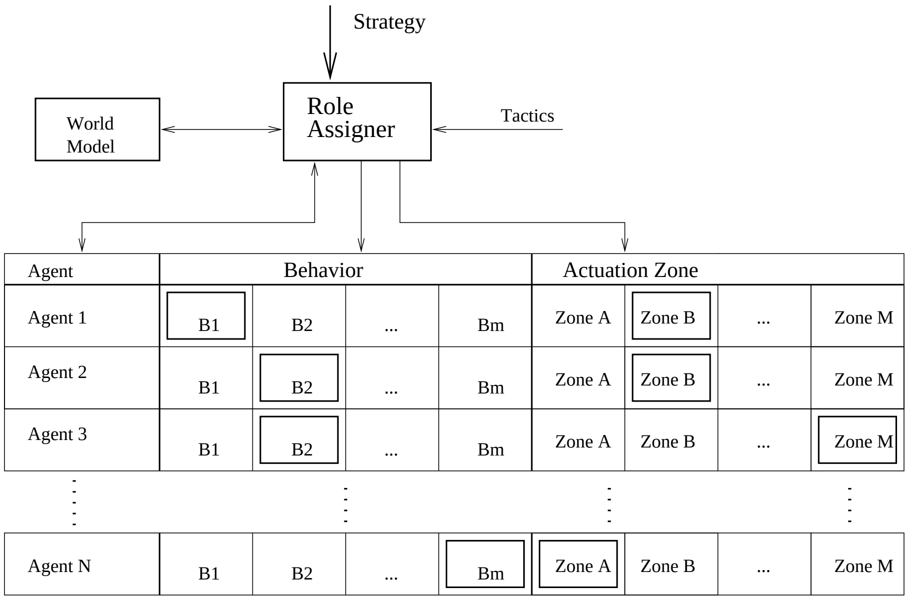


Figure 3.3: Relational Level – The highlighted boxes represent the active role (i.e., behavior and actuation zone) for an agent.

Tactics	Roles
Tactic_1	`Role1, Role2, Role2, Role3`
Tactic_2	`Role1, Role2, Role3, Role3`
Tactic_3	`Role2, Role2, Role3, Role4`
Tactic_4	`Role1, Role2, Role4, Role4`

Table 3.1: Example of mapping between Tactics and Roles

Strategy	Roles
Defend	`GoalKeeper, Defender, Defender, Midfieled`
Attack	`Defender, Midfield, Forward, Forward`
Block	`GoalKeeper, Defender, Defender, Defender`
Counter-Attack	`GoalKeeper, Defender, Midfield, Forward`

Table 3.2: Possible mappings between Strategies and Roles in the soccer domain, representing different tactics.

a given tactics (assuming a team composed by four robots). For a given *tactics*, the list of roles may be defined as the *team structure*. The list is ordered from left to right, the most important *role*, i.e., the one that is to implemented in the first place is the one on the left.

Table 3.2 presents a possible mapping between Tactics and Roles in the soccer domain. The role ordering for each tactic reflects the fact that the team should not suffer goals. Depending on the case, not suffering goals may be more important than scoring goals. With this mapping, the team will first assign the GoalKeeper role. At any point in time, should this robot have some malfunction, another robot would be re-assigned this *role*. The mapping between *role* and the pair (behavior, actuation zone) is detailed in the sequel.

The previously described operation can be described as a 3-tuple (Q, T, δ_r), where

Q : the set of strategies.

T : the set of pre-programmed roles;

δ_r is the mapping function;

The mapping δ_r is defined as a mapping between the prescribed strategy and the list of roles to be exhibited by the individual agents.

$$\delta_r : Q \rightarrow T$$

The role negotiation uses the *blackboard* as the communication mean. The *blackboard* stores, at the relational level, two ordered lists of roles. Each *role* in the list is composed of a pair, relating a *behavior* and a *position* (B,P). One of them, LD, is the ordered list that results from the tactical mapping. This represents the *desired team structure*.

$$LD = \{(B_1^d, P_1^d), (B_2^d, P_2^d), (B_3^d, P_3^d), (B_4^d, P_4^d)\} \tag{3.1}$$

LC is the *current team structure*, i.e., the list of active roles.

$$LC = \{(B_1^c, P_1^c), (B_2^c, P_2^c), (B_3^c, P_3^c), (B_4^c, P_4^c)\} \tag{3.2}$$

Over time the relational level of each robot compares LD with LC, using the function `DIFF`, where

DIFF is defined as the function that, given the lists LD and LC, determines the list of roles in LD that do not have a corresponding role in LC. Due to the fact that the list is ordered according to the role importance, the evaluation is done from left to right.

$$LDiff = DIFF(LD, LC), LDiff \subseteq LD \tag{3.3}$$

Another useful function to compare the desired and current *team structures* is the `CHANGE` function.

CHANGE is defined as the function that, given the lists LD and LC, determines the list of roles in LC that do not have a corresponding role in LD.

$$LChan = CHANGE(LD, LC), LChan \subseteq LC \tag{3.4}$$

Because roles are dynamic, robots may change (or not) their role during a game. Therefore, a function exists to compute the best role to assume, for a given robot, given a list of unassigned roles and the robot current role (MY_ROLE).

BEST is defined as the function that, given the list of unassigned roles $LDiff$ and the robot current role (MY_ROLE), determines the best unassigned role to be assigned to the robot, according to some domain-dependent criteria. It is to be applied if the robot current role is in $LChan$.

In the soccer domain, the criteria used to evaluate the best option is the distance between the robot current actuation zone and the unassigned roles actuation areas. This criteria avoids situations such as a player assigned to the front actuation area changing to the back area if another robot is unassigned and closer.

An algorithm presented was envisaged to be applied over time, at the relational level, in order to ensure that the team reconfigures itself according to the game dynamics. The included time out ($TIME\_OUT$) guarantees that if more that one robot applies for a given role simultaneously, only one of them is assigned to that role. That way, robots whose role is a surplus for the desired configuration re-enters the role reassignment process.

The algorithm, in pseudo code, follows:

```
Ldiff = DIFF(LD, LC)
LChan = CHANGE(LD,LC)
if LDiff = { }                                 #
  Keep_Role                                    # no changes are necessary
else
  if MY_ROLE ∈ LChan                           # may have to change
    NEW_ROLE = BEST (LDiff, MY_ROLE);          # determine best change
    TIME_OUT;                                  # random time out to
                                               # see if other picks it up
    if MY_ROLE ∈ CHANGE(LD, LC)                # still surplus
      MY_ROLE = NEW_ROLE                       # self assign new role
    else
      Keep_Role                                # keep current role
  else
    Keep_Role                                  # keep current role
```

3.2.3 Individual Level

In order to implement the individual level of the functional architecture described in Section 3.1, the architecture shown in Figure 3.4 is implemented in each robot of the population. The *Behavior Coordinator* (top layer), selects the correct behavior for the robot, based on the *role* assigned by the relational level. When a behavior is selected, a corresponding *spatial supervisor* is also activated. Different behaviors are associated to different influence areas, corresponding to working areas. The *spatial supervisor* ensures that the robot always stays inside its assigned actuation zone.

In this context, a behavior is defined as follows:

Definition 4 *A* behavior *corresponds to a set of purposive (i.e., with a goal) states, sequentially and/or concurrently executed.*

The state transitions may be described by the 3-tuple (P, C, δ_b), where the symbols have the following meanings:

P : the set of primitive tasks (state)

C : the set of logical conditions

δ_b : the *next_state* function

The δ_b function is defined on the cross product between the set of primitive tasks and the logical conditions, mapped to the set of primitive tasks.

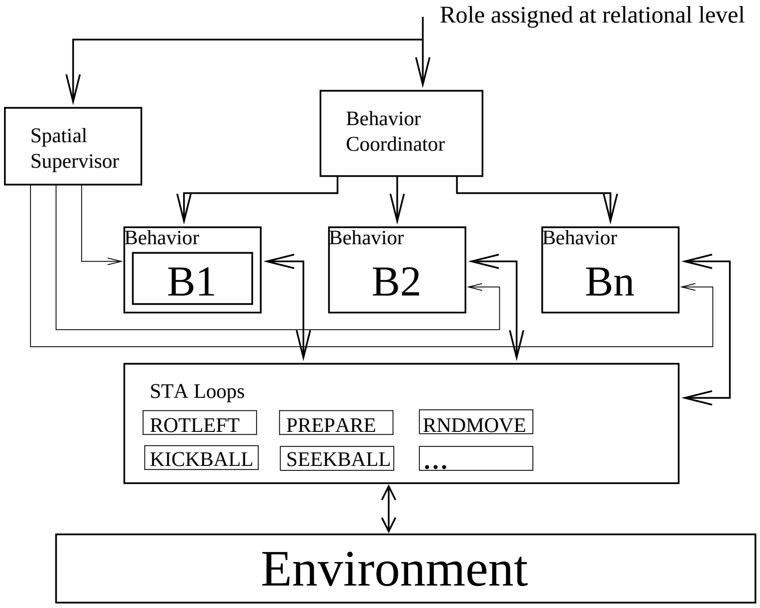


Figure 3.4: Individual Level.

$$\delta_b : P \times C \to P$$

Where *logical condition* is defined over a predicate set and check the value of a variable or a past event (both types are stored in the world model) (e.g., `see(ball) & near(ball)` ; `lastseen(ball)=right`);

A *primitive task* consists of *sense-think-act loops* (STA loops). Those are a generalization of a closed loop control system which may include, motor control, ball tracking or trajectory following control loops. They are composed of the following key components:

- **goal**: the objective to be accomplished by the primitive task (e.g., a position set-point, an object to be found in the image, location with respect to an object in the image);
- **sense**: sensor data required to accomplish the goal (e.g., distance to an object, object position in an image);
- **think**: the actual algorithm which, using the sensor data, does what is required to accomplish the goal (e.g., motion controller, ball visual servoing);
- **act**: the actions associated to the **think** algorithm (e.g., moving the wheel motors).

STA loops provide the behavior designer with a set of high-level functionalities to use. Those are closely connected to the system hardware as they are built upon the set of *system functions* (e.g., ROTLEFT, KICKBALL, PREPARE, SEEKBALL).

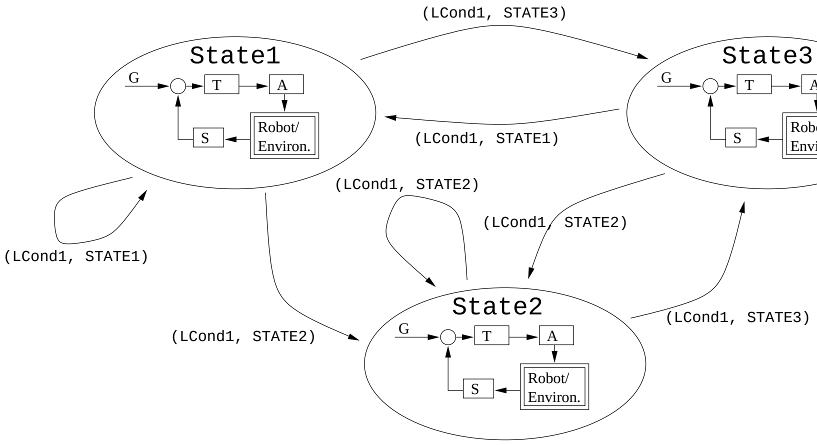


Figure 3.5: Behavior State Machine.

3.2.4 Data Distribution

In order to implement the proposed architecture, data must traverse its functional levels. Following the principle that data should be close to the unit that processes it, data is distributed through the population.

A distributed *blackboard* was implemented to allow global memory sharing and event-based communications [33]. The *blackboard* is the single medium of communication between the agents, supporting the message exchange required for negotiation. One of the key factors is the distribution of data per robot, which minimizes the need to communicate in order to obtain information (e.g., data obtained from processing an image should be stored in the robot where the image was acquired). The global team strategy and tactics are also stated in the *blackboard*.

Besides the physical distribution, a functional distribution was also envisaged. The data is stored at different levels of abstraction, according to the functional levels that use it. For example, in order to decide the strategy to be followed, the *organizational level* does not need to know the agents exact position or if any of them is colliding with another. Instead, it uses information such as *list of active roles* and *task duration*.

Chapter 4

CASE STUDY – ROBOTIC SOCCER

This chapter presents the case study used to illustrate the architecture presented in Chapter 3: robotic soccer. The problem is introduced, and the implementation details on *strategies*, *tactics*, *roles*, *behaviors* and *primitive tasks* regarding this application are given. A brief discussion on the implementation is also presented.

4.1 The RoboCup Challenge

RoboCup [53] [30] is the Robot Soccer World Cup. Organized on an annual basis by the RoboCup International Federation, it is a contest that aims at fostering intelligent robotics research by providing a standard problem where a wide range of technologies can be integrated and examined. It has two particularities that make it very interesting:

- The game is *extremely dynamic*. Robots have to deal with moving obstacles, take control of a ball, avoid opponents (which make unpredictable movements) and even have to score goals.

- *Real robots* have to be used in the competitions (except for the simulation league). Those must be robust and reliable.

The design of a team to participate in RoboCup includes design principles of autonomous agents, multi-agent cooperation, strategy acquisition, real-time reasoning, robotics, sensor-fusion, project management and logistics.

The contest and its challenges, has served as a motivation and a inspiration to the case study presented in this chapter. Part of the system was tested in the

1998 edition of RoboCup, held in Paris [5] [71] [70] and in the 1999 edition, held in Stockholm [69] [68].

4.1.1 Game Concepts

Soccer is a game played by two opposing teams, moving in a rectangular area (*field*) (see Figure 4.1). Each team is assigned half field, which includes a goal in the corresponding half field top. The objective is to score goals in the opponent goal. Rules exist to control the game flow.

When a team controls the ball and is moving in the opponents goal direction, it is *attacking*. Players comfortable in attack positions are called *Forwards* because they actuate near the opponent goal. If the team is not in control of the ball and the majority of its players is in the team own field, it is *defending*, as it waits for the other team to attack. *Defenders* are the players that stay closest to the team own goal. Should a team be in defense mode and got control of the ball, it should change to *counter_attack*, in order to move the ball to the opponent field and then change to *attack*. Players that stay between the teams defense zones are called the *Midfielders*. Those players are in good position to quickly move to attack or defense if necessary.

4.1.2 Soccer Robotics

The Robocup challenge has been designed in accordance with FIFA rules [54]. This option was taken in order to make it more realistic and to allow a real convergence with human soccer in a time limit of 50 years.

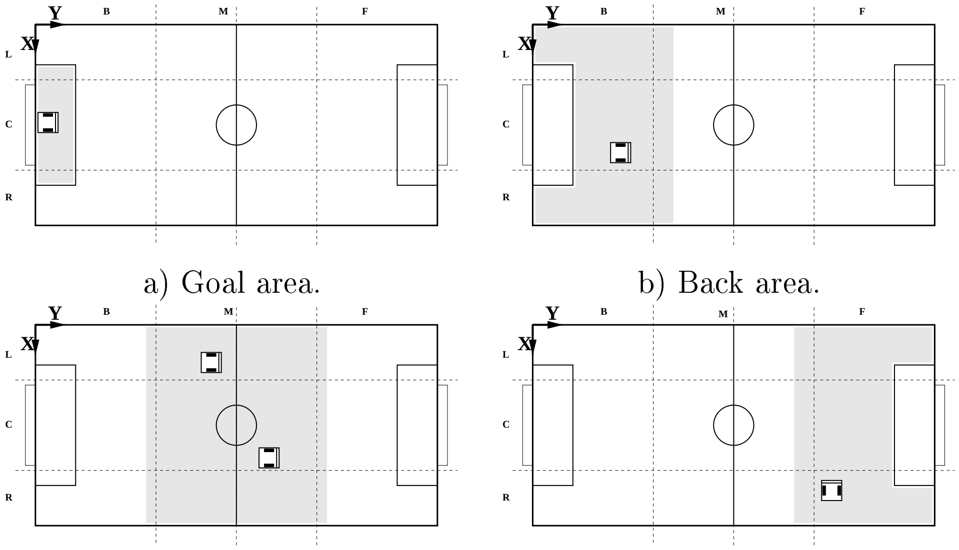
Besides the written rules, there are a number of *thumb rules* that one should consider when designing a robotic soccer team. Examples of this knowledge follow:

- Defense should not be left open, i.e., with too few players.
- Players should not get too close in order to cover a broader area.
- The Goalkeeper must not leave the goal area for a long time.

When designing tactics, this kind of knowledge is essential.

4.1.3 Field Division

Figure 4.1 presents a functional division of the soccer field in several regions. Those zones are used to easily locate the robots inside the field, according to their assigned behaviors.



a) Goal area. b) Back area.

c) Middle area. d) Front area.

Figure 4.1: Actuation Zones.

Besides Back *(B)*, Middle *(M)* and Front *(F)*, the field is divided in Left *(L)*, Center *(C)* and Right *(R)* zones to increase resolution. This division is particularly useful when the team has more than one player acting in the same functional area (e.g., two defenders). In that case, the players should negotiate an auxiliary division for repositioning. In the sequel, the Middle area is further divided in Middle Back *(MB)* and Middle Front *(MF)* in order to increase the field resolution.

4.2 Soccer Individual Level Roles

As stated before, soccer is a complex game where players have precise tasks, assigned according to their capabilities. In order to implement a team of soccer playing robots, the following roles were assumed:

- GoalKeeper.
- Defender.
- MidFielder.
- Forward.

Two additional roles were defined to handle special situations.

- Penalty Shooter.

- Foul kicker.

In the following subsections the designed behaviors are detailed. The STA loops used in their implementation are also described. The initial behaviors were designed without low-level constraints. After their design, the required *primitive tasks* were designed to allow the implementation of the behaviors. In case a required *primitive task* could not be implemented, the behaviors would be modified and a new set of *primitive tasks* designed. This design process was iterated until convergence was achieved.

4.2.1 Behaviors

Five behaviors to be used in the role definition were identified and implemented:

- **LOOK** (L)
- **LOOK-KICK** (LK)
- **LOOK-PUSH** (LP)
- **FOLLOW-KICK** (FK)
- **FOLLOW-PUSH** (FP)

LOOK

This behavior makes the robot rotate around is vertical axis, looking for the ball. No translational movement exist. The proposed state machine to implement it is presented in Figure 4.2. This behavior includes the *primitive tasks* `SEEKBALL`, `ROTLEFT` and `ROTRIGHT`. Those are presented and detailed in the sequel.

When not seeing the ball, the robot rotates in the direction were the ball was seen for the last time. It rotates continuously until the ball is visible again. If the ball is visible, the robot begins tracking it.

Using the formalism depicted in Section 3.2.3, the *LOOK Behavior* is defined as follows:

P : { ROTLEFT, ROTRIGHT, SEEKBALL }

C : { see(ball), lastseen(ball)=left, lastseen(ball)=right }

δ_b is defined in Table 4.1.

All the cases that are not listed in Table 4.1 are ignored, which means that the state is kept unchanged (e.g. ROTLEFT $\times \sim$ see(ball) $\rightarrow$ ROTLEFT).

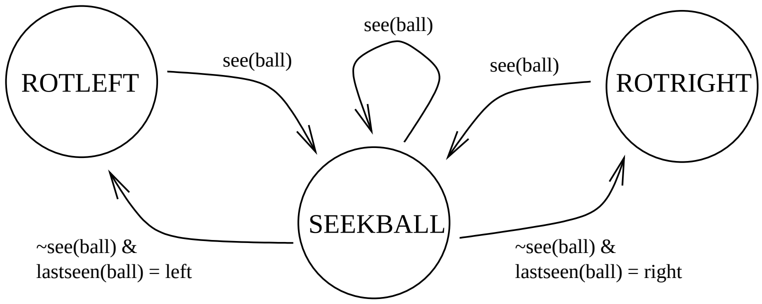


Figure 4.2: LOOK Behavior.

P	C	$\delta_b : P \times C \rightarrow P$
ROTLEFT	see(ball)	SEEKBALL
SEEKBALL	see(ball)	SEEKBALL
SEEKBALL	$\sim$ see(ball) & lastseen(ball) = left	ROTLEFT
SEEKBALL	$\sim$ see(ball) & lastseen(ball) = right	ROTRIGHT
ROTRIGHT	see(ball)	SEEKBALL

Table 4.1: δ_b mapping

LOOK-KICK

This behavior makes the robot rotate around its vertical axis, looking for the ball. If the ball comes closer than a given distance, the robot kicks it away. The proposed state machine to this behavior it is presented in Figure 4.3. This behavior includes the *primitive tasks* `SEEKBALL`, `ROTLEFT`, `ROTRIGHT` and `GKICKBALL`. Those are presented and detailed in Section 4.3.

LOOK-PUSH

Like LOOK-KICK, LOOK-PUSH behavior makes the robot rotate around its vertical axis, looking for the ball. The difference is that when the ball comes closer than a given distance, it is pushed, i.e., the robot moves towards the ball, whit out kicking. This movement allows the robot to control the ball until a kicking situation arises. The proposed state machine to implement it is presented in Figure 4.4. This behavior includes the *primitive tasks* `SEEKBALL`, `ROTLEFT`, `ROTRIGHT` and `GPUSHBALL`. Those are presented and detailed in Section 4.3.

The single difference between this behavior and the previous ones is the force given to the ball. In this case the ball is not kicked, which allows for a greater

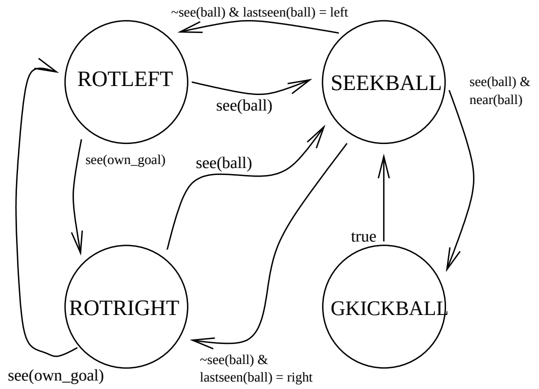


Figure 4.3: LOOK-KICK Behavior.

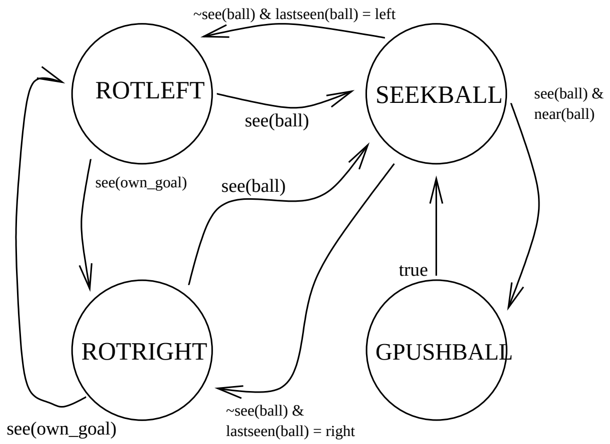


Figure 4.4: LOOK-PUSH Behavior.

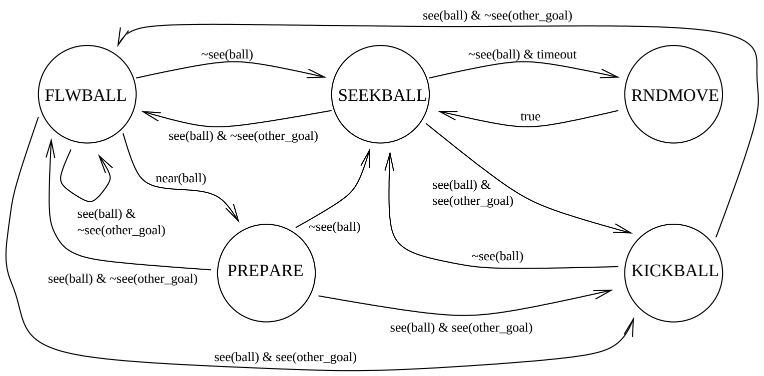


Figure 4.5: FOLLOW-KICK Behavior.

control and can be used to pass it to a team mate.

The need for this behavior was learned during the Robocup98 edition. In some situations, when the goalkeeper kicked the ball, it would hit some opponent player and would return to our goal. When the ball was not kicked but pushed, the control was better and the goalkeeper would return to its original position before another attack was performed.

FOLLOW-KICK

The FOLLOW-KICK behavior starts by making the robot look for the ball (SEEKBALL). When the ball is found, the robot moves towards it (FLWBALL). At a given distance, if the alignment with the ball and the opponent goal is not good enough, the robot actively tries to re-position itself in order to have the ball between itself and the opponents goal. Up to this moment, this maneuver is done is open loop, which leads to occasional positioning failures. When that configuration is reached and the ball is close enough, it kicks the ball. The proposed state machine to implement it is presented in Figure 4.5. This behavior includes the *primitive tasks* `FLWBALL`, `SEEKBALL`, `RNDMOVE`, `PREPARE` and `KICKBALL`. Those are presented and detailed in Section 4.3.

FOLLOW-PUSH

FOLLOW-PUSH behavior is similar to the FOLLOW-KICK behavior, like the LOOK-PUSH/KICK behavior differs from FOLLOW-KICK behavior in the kick part. Here the ball is not kicked away but kept close to the robot. The proposed state machine to implement it is presented in Figure 4.6. This behavior includes the *primitive tasks* `FLWBALL`, `SEEKBALL`, `RNDMOVE`, `PREPARE` and `PUSHBALL`. Those are presented and detailed in Section 4.3.

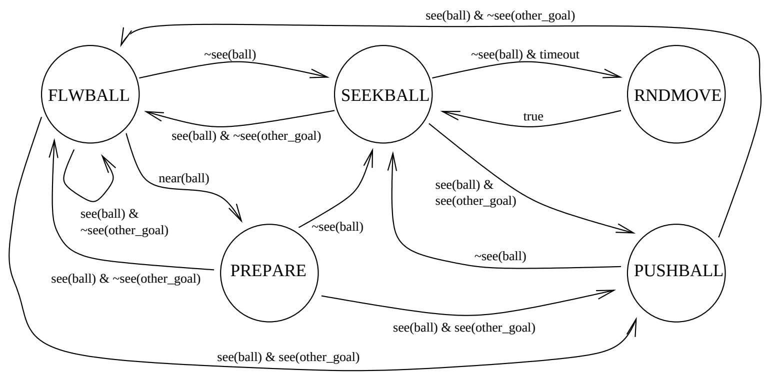


Figure 4.6: FOLLOW-PUSH Behavior.

With this behavior it is possible to control the ball and move it close to the robot in the opponents goal.

4.2.2 Roles

GoalKeeper

The goalkeeper mission is to prevent the ball from entering the goal. To do that, it must continuously look for the ball and, if necessary, leave its position and kick or push it away. Its actuation area is defined by the goal area lines (G in Figure 4.1 a)). To implement the GoalKeeper, the LOOK-KICK or the LOOK-PUSH behaviors are used.

Defender

The defender primary mission is to move the ball from the vicinity of its team back area to the opponents field. Its actuation zone is depicted in Figure 4.1 b). Another possibility is to block balls from moving close to the goal area. To implement this role, one of two behaviors can be used. LOOK-KICK should be used if the player is to wait for the ball to get into its area and then kick it away. This can be useful if there are more players assigned to the same role, in order to behave like a wall. Alternatively, FOLLOW-KICK gives the player more freedom to move in its actuation area. With that behavior, it will go for the ball and than kick it in the opponent goals direction. If few players are assigned the defender role, this behavior should be used because it allows for a better area coverage.

MidFielder

Like in real soccer, the MidFielder actuation area lies within both half fields (see Figure 4.1 c)). This player natural ability is to receive the ball from its own team field and move it to the opponents field, in the direction of its goal (if possible). On the other way, if the other team is controlling the ball and is moving into the midfield half field, it can be used like a barrier, preventing the ball from approaching its goal. The usual behavior to use is FOLLOW-PUSH because it will move the ball in the opponents area in a controlled way, making it easier for the forward players to take control of it. An alternative behavior to use would be LOOK-KICK. That would make the player act as a barrier. If other players were assigned similar roles, a very effective line may be built at midfield.

Forward

The Forward behavior induces the player to be in the F zone (see Figure 4.1 d)). If the ball goes into the team field, the F players should keep track of the ball. When the ball moves into the F zone, it must try to take control over it as soon as possible and kick it into the opponents goal. FOLLOW-KICK is the used behavior. Should the strategy favor counter attack and the appropriate tactic exist, the forward role could be assigned the LOOK-KICK behavior while the ball is not close (to keep trace of it) and than change to the FOLLOW-KICK in order to approach the ball, control it and score a goal.

4.3 Primitive Tasks

The primitive tasks are the *building blocks* for the behaviors. In the sequel, the list of primitive tasks implemented and available to built behaviors are presented and explained.

The examples refer to an implementation on a robot equipped with differential drive mobility structure (described in Chapter 5 and detailed in Appendix A), using a camera pointing ahead, making a small angle with the ground plane. In the following, vc represents the robot longitudinal velocity and vd is the differential velocity. W is the image width, xc is the center of mass and $DSCALE$ is an adjustable gain.

SEEKBALL

- **goal**: Keep the ball image in the center of the captured image (`XREF = 0`);
- **sense**: Determine ball center of mass in the image (`GetBall(img)`);
- **think**: Compute the speed set-points (e.g., `vd = (xc - XREF) / DSCALE; vc = 0`);

- **act**: Move the robot (e.g., `motSetVel( vc + vd, vc - vd )`);

SEEKBALL is intended to keep the robot heading at the ball. It moves the robot to keep the ball center of mass in the image coincident with the image center (along x). The result is a rotation around the vehicle axis. It is useful to search for the ball.

FLWBALL

- **goal**: Keep the ball center of mass in the center of the captured image (`XREF = 0`) and approach it;
- **sense**: Determine ball position in image (`GetBall(img)`);
- **think**: Compute the speed set-points (`vd = (xc - W/2` $\pm$ `W/4 ) / DSCALE; vc = VTRANS`);
- **act**: Move the robot (e.g., `motSetVel( vc + vd, vc - vd )`);

FLWBALL is similar to SEEKBALL. The difference is that FLWBALL looks for the ball while moving. For instance, it may be used by a forward player to move the ball into the goal area or by a defense to move the ball into the opponent field.

ROTRIGHT

- **goal**: Make the vehicle continuously rotate to the right;
- **sense**: N/A;
- **think**: `vc = 0;vd = VROT`;
- **act**: Move the robot (`motSetVel( vc + vd, vc - vd )`);

ROTRIGHT continuously rotates the vehicle to the right.

ROTLEFT

- **goal**: Make the vehicle continuously rotate to the left;
- **sense**: N/A;
- **think**: `vc = 0;vd = -VROT;`;
- **act**: Move the robot (`motSetVel( vc + vd, vc - vd )`);

ROTLEFT makes the vehicle continuously rotate to the left.

GKICKBALL

- **goal**: Kick the ball and return to origin;
- **sense**: N/A;
- **think**: Set the kick time and speed and the kicker (t_{kick} `= TKICK; vc = VKICK; kick = ON`);
- **act**: Move the robot (`bb.timer = 0`; `while (bb.timer < TKICK) motSetVel(vc, vc)`; Set the kicker (`setKick(kick)`);`bb.timer = 0`; `while (bb.timer < TKICK) motSetVel(-vc, -vc)`; `motsetvel( 0, 0)`);

This primitive task is used to move the robot against the ball. It enables the kicking mechanism to hit hard the ball when it touches the robot. After a given time (t_{kick}) the robot is moved back to position it was before the kick.

KICKBALL

- **goal**: Kick the ball;
- **sense**: N/A;
- **think**: Set the kick speed and the kicker on (`vc = VKICK; vd = 0; kick = ON`);
- **act**: Move the robot (`motSetVel( vc + vd, vc - vd )`); Set the kicker (`setKick(kick)`);

This primitive task is used to move the robot against the ball. It enables the kicking mechanism to hit hard on the ball when it touches the robot. Contrary to the GKICKBALL primitive task, this one does not reverse motion after the kick.

GPUSHBALL

- **goal**: Push the ball and return to origin;
- **sense**: N/A;
- **think**: Set the push time and speed and turn off the kicker (t_{push} `= TPUSH; vc = VPUSH; kick = OFF`);
- **act**: Move the robot (`bb.timer = 0`; `while (bb.timer < TPUSH) motSetVel(vc, vc)`; `bb.timer = 0`; `while (bb.timer < TPUSH) motSetVel(-vc, -vc)`; motsetvel(0, 0));

GPUSHBALL is used to move the ball, close to the robot, in a controlled way. It can be used to move the ball away from the goal area without kicking it, in order to avoid reflections in other robots. It does not enables the kicking mechanism. After the a given time pushing, the robot is moved back to its original positions.

PUSHBALL

- **goal**: Push the ball;
- **sense**: N/A;
- **think**: Set the push speed and turn off the kicker (`vc = VPUSH; vd = 0; kick = OFF`);
- **act**: Move the robot (`motSetVel( vc + vd, vc - vd )`);

PUSHBALL is used to move the ball, close to the robot, in a controlled way. It can be used to prepare a kick, while aligning. It does not enables the kicking mechanism.

RNDMOVE

- **goal**: Move the robot in a random direction and velocity;
- **sense**: Get a random speed (`speed = GetRandSpeed()`);
- **think**: Set the speed set-points (`vd = sign(speed)*VRAND; vc = VRAND;`);
- **act**: Move the robot (`motSetVel( vc + vd, vc - vd )`);

RNDMOVE makes the robot move in a random way. It generates a random number that uses to control the speed and direction of movement. It is done in low speed.

PREPARE

Makes a pre-programmed movement in order to circulate some obstacle.

- **goal**: Position the robot so that the ball is between it and the opponent goal;
- **sense**: N/A;
- **think**: Set the speed set-points (`vd1=VD1; vc1=VP1; vd2=VD1; vc2=VP1; vd3=VD3; vc3=VP3`);
- **act**: Move the robot (`bb.timer = 0`; `while (bb.timer < TP1) motSetVel(vc1+vd1, vc1-vd1)`; `while (bb.timer < TP2) motSetVel(vc2+vd2, vc2-vd2)`; `while (bb.timer < TP3) motSetVel(vc3+vd3, vc3-vd3)`);

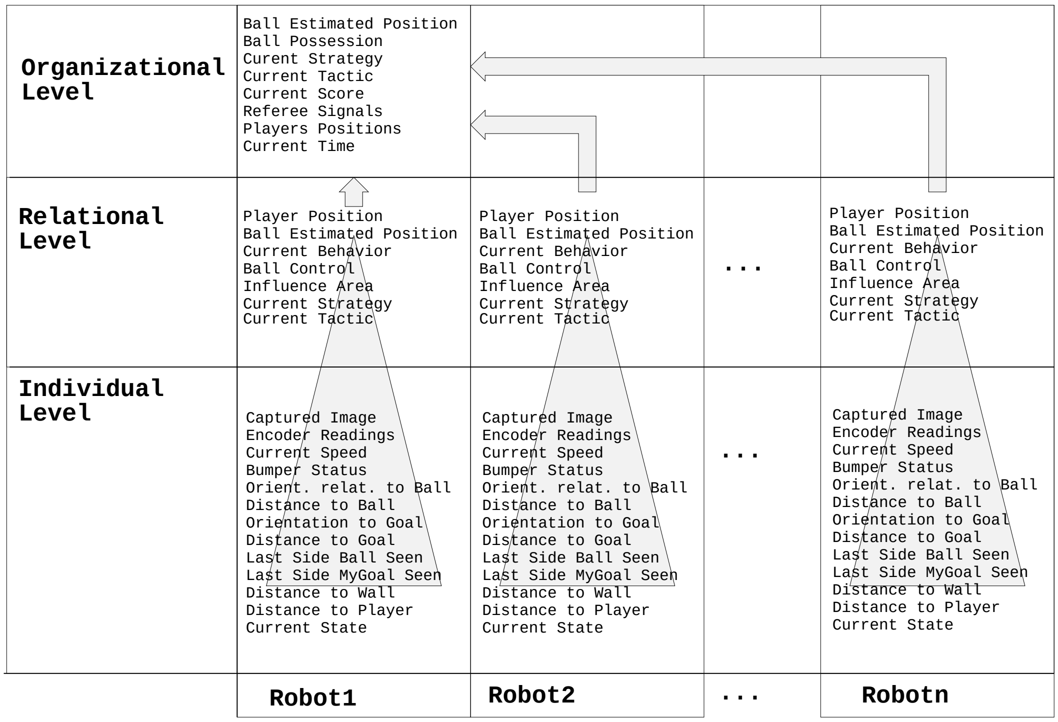


Figure 4.7: Sensorial data flow through the system.

4.4 Data Distribution

Data should stay close to the processes that use it most of the time, therefore, all team data is distributed over the robots. This concept was chosen because it increases the system robustness, i.e., if one of the robots has a malfunction, the team will continue to work. In addition, it also simplifies the process of adding agents to the team and speeds data processing.

Figure 4.7 presents a schematic view of the data flow through the different control levels and robots. The order in which the data items appear is not related to its importance. Note that all the *organizational level data* is only stored in one of the robots. This comes from the fact that this level is active in *only one* of the robots at a single time. It acts like a team captain in real soccer. Should this particular player have a malfunction and stop, another team player would rise as captain (chosen by computational performance benchmarking). In such a case, it would have to gather again the *organizational level* information from scratch, i.e., the data stored in the stopped player would be lost.

Figure 4.8 presents the different processes running simultaneously in each robot and the data flow between them. The arrows represent the data dependencies that exist between processes. For example, the `state machine` process needs information from the *sensors* (e.g., an image) and from the *vision* process (e.g., ball position in

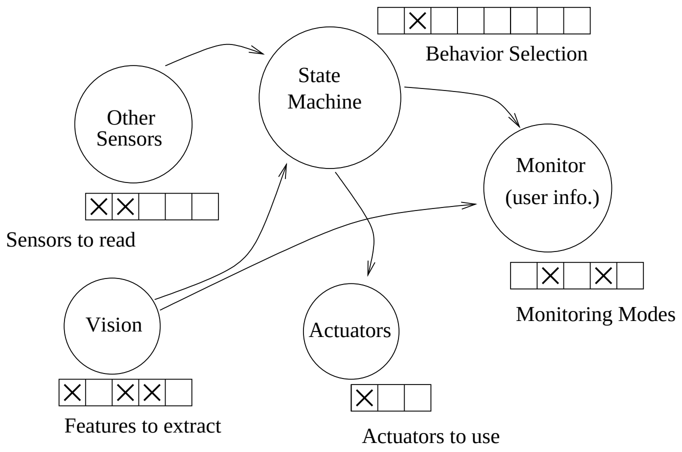


Figure 4.8: Processes and data flow.

image) in order to work.

Note that all the information gathered by the different processes is stored in a common *blackboard*, accessible by all processes [66], not represented in Figure 4.8. The crossed boxes shown next to the different modules represent a set of `FLAGS` associated to each process. Depending on the task being executed, a given information may not be necessary and therefore, there is no need to acquire and process it. This mechanism aims at reducing the computation time. A *proxy* process that handles communications is running but is not shown in the figure.

4.5 Organizational Level Implementation

The organizational level is a high level controller of the team. Its input is the current world state and its output a given strategy to be followed by the team in order to attain a long term goal. Currently the goal is defined at design time, being embodied in the available strategies, tactics and roles.

This level is implemented by a two level state machine. This comes from the fact that, in the soccer domain, the game state belongs to one of two classes: **game situations** and **game status**, where,

Game situations are reached upon the evaluation of tournament rules (e.g., kick-off, fault, end-of-the-game, penalty).

Game status translates the current status (e.g., score, elapsed-time, ball-position, ball-possession).

This division is mapped in two strategy categories:

Pre-programmed scenarios for game situations, such as `kickoff`, `penalty-for`.

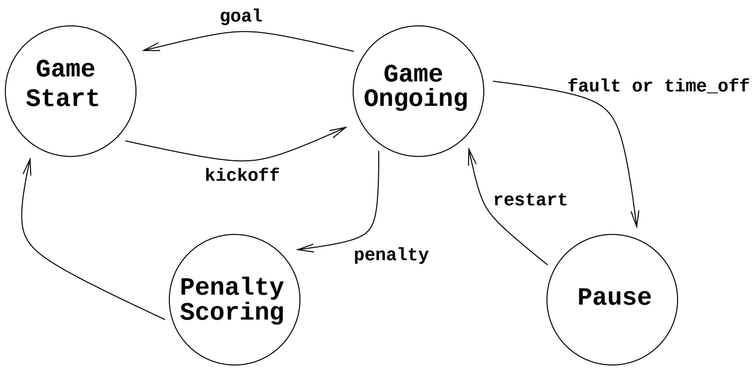


Figure 4.9: Game situations state machine.

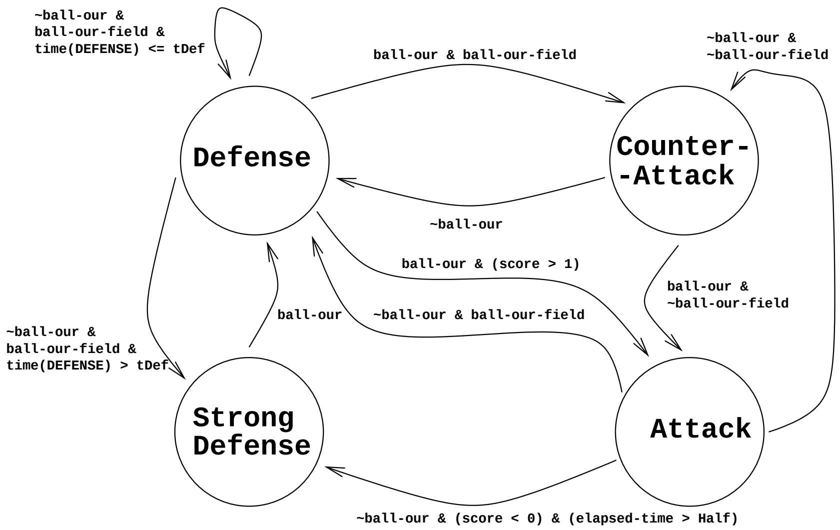


Figure 4.10: Game status state machine, corresponding to the *Game Ongoing* in Figure 4.9

Dynamic strategies correspond to the on-going game. This includes strategies such as `defend`, `counter-attack` and `attack`, which evolve depending on the game status.

A proposed *game situations* state machine is presented in Figure 4.9. The prescriptions issued by this level overlap the ones prescribed by the *game status* state machine. When the game kickoff is signaled, the *Game Ongoing* state is reached. This is a special state that does not prescribe a strategy by itself. When in this state, the strategy is prescribed by the current state in the *game status* state machine. During the game, in case of fault or if the game final is reached, the state is changed to *pause* which causes the strategy *pause* to be prescribed. Another situation that may stop the game is a *penalty*.

Figure 4.10 presents a proposed *game status* state machine whose states are traversed upon the game dynamics. The transitions between *game status* states are

Q	L	$\delta_s : Q \times L \to Q$
Defense	∼ ball_our & ball_our_field & time(Defense) <= tDEF	Defense
Defense	∼ ball_our & ball_our_field & time(Defense) > tDEF	Strong-Defense
Defense	ball_our & ball_our_field &	Counter-Attack
Defense	ball_our & score > 1	Attack
Strong-Defense	ball_our	Defense
Counter-Attack	∼ ball_our	Defense
Counter-Attack	ball_our & ∼ ball_our_field	Attack
Attack	∼ ball_our & ∼ ball_our_field	Counter-Attack
Attack	∼ ball_our & ball_our_field	Defense
Attack	∼ ball_our & score < 0 & elapsed-time > Half	Strong-Defense

Table 4.2: δ_s mapping

traversed according to the game dynamics and the pre-programmed strategy.

Using the notation defined in Section 3.2.1 the *Game Status* state machine can be described as follows:

Q : { Defense, Strong-Defense, Counter-Attack, Attack}

S_c : { Defense }

L : { ball_our, ball_our_field , score, elapsed-time, time}

δ_s is defined in Table 4.2. All the cases that are not listed in the table are ignored, which means that the state is kept unchanged.

4.6 Relational Level – From Strategy to Roles

At the relational level a translation is made between the prescribed strategy and players *roles*. Roles are assigned to the different agents, according to the pre-defined *tactics* (the mapping between strategies and roles). Different tactics may be defined for the same strategy.

- game-start – In a game start situation the players reach their pre-determined start positions (see Figure 4.11). Precise positioning of players is accomplished at this state as they must be correctly positioned prior to game start. When

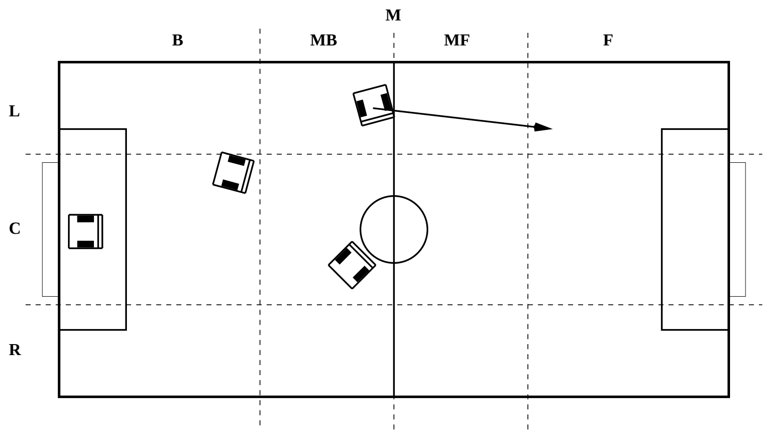


Figure 4.11: Players position at game start-up.

Pause	L,-	L,-	L,-	L,-

Table 4.3: Pause Tactics

positioned, the players wait for the kickoff external signal (e.g., a whistle blew) that starts the game.

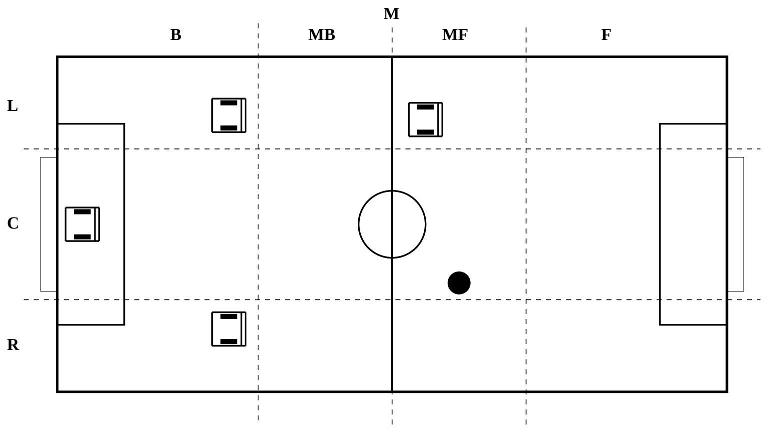
- pause – A game pause arises if a fault is signaled or the game end is reached. All players should immediately stop. The defined tactics is presented in Table 4.3. It makes the robots stop while looking for the ball. No translational movement is done.

- defense – The ball must be prevented from entering our field. If it enters, it must be moved into the opponents field and ultimately, into its goal. Several defense tactics may be defined. The one to use is chosen based on some criteria, dependent on the game state.

  The defense tactics is presented in Figure 4.12. Besides the Goalkeeper, two defenders and a Midfielder are used. Their behaviors are basically LOOK-PUSH because the players are to hold their positions.

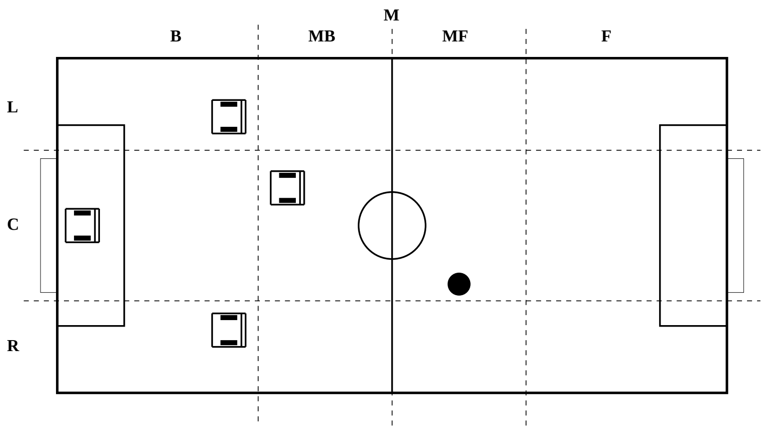
- strong-defense – Points towards a strong defense and a good recovery mechanism, essential for counter-attack. The concept is illustrated in Figure 4.13. The difference between the strong-defense and the defense is that in defense not all players are moved into our field. This gives the possibility of making the transition to counter-attack easier, as one of the players stays in the opponent field.

  The strong defense tactics is presented in Figure 4.13. Besides the Goalkeeper,



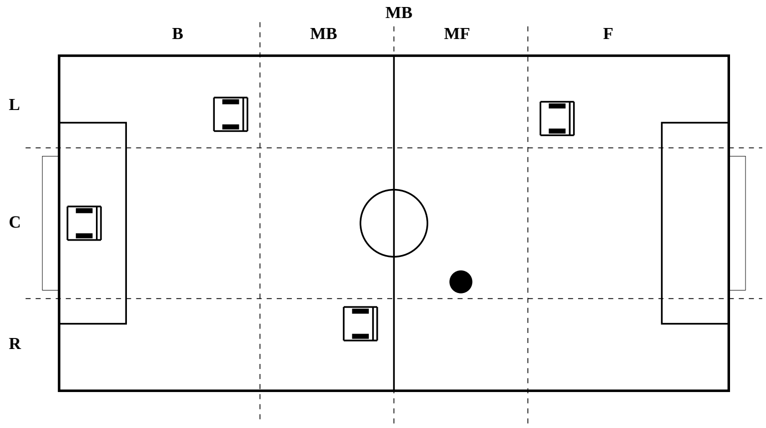
Defense	LK,G	LP,BL	LP,BR	LK,M

Figure 4.12: Example of Defense Tactics.



Strong Defense	LK,G	LP,BL	LP,BR	LP,MBC

Figure 4.13: Example of Strong defense Tactics.



Counter Attack	LK,G	FP,B	FK,M	L,F

Figure 4.14: Example of Counter attack Tactics.

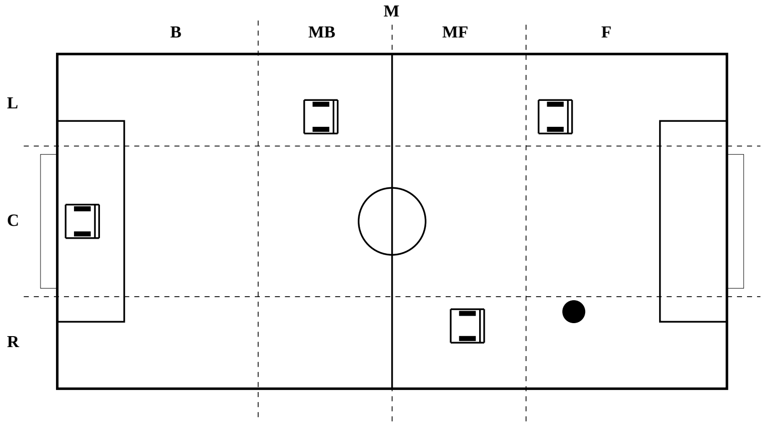
three defenders are used. Their behaviors are basically LOOK-PUSH because the players are to hold their positions.

- counter-attack – A counter-attack happens if the team is positioned to move the ball quickly into the opponent field and score a goal. It requires a Defender, to prevent a possible intersection of the ball by an opponent, a Medium, to pass the ball into the Forward area, and a Forward player to kick the ball into the opponent goal. If the movement is performed without major opposition (i.e., it does not start moving towards our field), it should be changed to an attack movement. The counter-attack scheme is presented in Figure 4.14.

- attack – In an attack movement all team is moved to the front. Besides the GK, there is only one player in our field (see Figure 4.15). This movement requires two MidField players (one in the MB zone and another in MF zone), and one forward. The idea is to have the ball passed from the M zone into the F zone, where a Forward player is supposed to pick it up and kick it at the goal.

A mapping summary is presented in Table 4.4.

An example of the application of the algorithm presented in Section 3.2.2 is presented in Table 4.5. The presented case refers to a change from *Defense* to *Strong Defense*. The `BEST` function used was the minimal distance between the current and the desired actuation zones. The `TIME_OUT` is not shown because only one robot is changing role.



Attack	LK,G	FK,MB	FP,MF	FK,F

Figure 4.15: Example of Close attack on the opponents goal Tactics.

Strategy	Role	Role	Role	Role
Defense	LK,G	LP,BL	LP,BR	LK,M
Strong Defense	LK,G	LP,BL	LP,BR	LP,MBC
Counter Attack	LK,G	FP,B	FK,M	L,F
Attack	LK,G	FK,MB	FP,MF	FK,F
Penality For	LK,G	L,B	L,B	FK,M
Penality Against	LK,G	L,M	L,M	L,M
Game Start	LK,G	LK,B	LK,MBL	LK,MC
Pause	L,-	L,-	L,-	L,-

Table 4.4: Different Mappings between Strategies and Roles (i.e., different Tactics)

n	Tactics	Status	Robot1	Robot2	Robot3	Robot4
0	Defense	LD={(LK,G) (LP,BL) (LP,BR) (LK,M)} LC={(LK,G) (LP,BL) (LP,BR) (LK,M)} LDiff= { }	LK,G Keep_Role	LP,BL Keep_Role	LP,BR Keep_Role	LK,M Keep_Role
1	Strong Defense	LD={(LK,G) (LP,BL) (LP,BR) (LP,MBC)} LC={(LK,G) (LP,BL) (LP,BR) (LK,M)} LDiff= {(LP,MBC)}, LChan = {(LK,M)}	LK,G Keep_Role	LP,BL Keep_Role	LP,BR Keep_Role	LK,M MY_ROLE $\in$ LChan BEST(LDiff, (LK,M))= LP,MBC NEW_ROLE = BEST(LDiff, (LK,M)) **MY_ROLE = LP,MBC**
2	Strong Defense	LD={(LK,G) (LP,BL) (LP,BR) (LP,MBC)} LC={(LK,G) (LP,BL) (LP,BR) (LP,MBC)} LDiff= { }	LK,G Keep_Role	LP,BL Keep_Role	LP,BR Keep_Role	LP,MBC Keep_Role

Table 4.5: Example Role Change During Game

Chapter 5

EXPERIMENTAL TESTBED

In this chapter, the hardware developed to build the testbed used to test the concepts introduced in Chapter 3 and 4 is presented. It consists of a population of mobile autonomous robots and a soccer robot field. Construction details are presented in the Appendices. Results obtained using the Testbed are also presented and discussed in this chapter.

5.1 Robotic Population

The robots composing the population must cope with the functional architecture depicted in Chapter 3. Namely, they must have enough computational power to implement the different levels of competence, autonomy regarding power, communication capabilities, ability to recognize other objects and robots and ideally, the capability of self-localizing in a known world. In the soccer domain, the kicking ability is also valuable. Those functionalities served as a guide to the implementation of the different modules.

The developed robots are not hardwired to play robotic soccer. They may be used for other applications and were built as an open architecture with this objective.

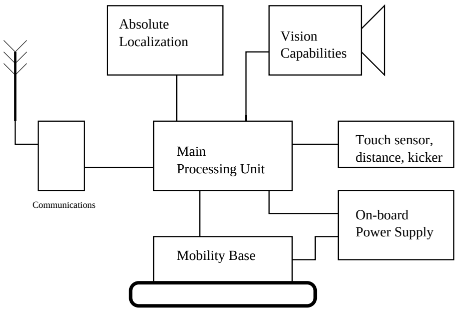
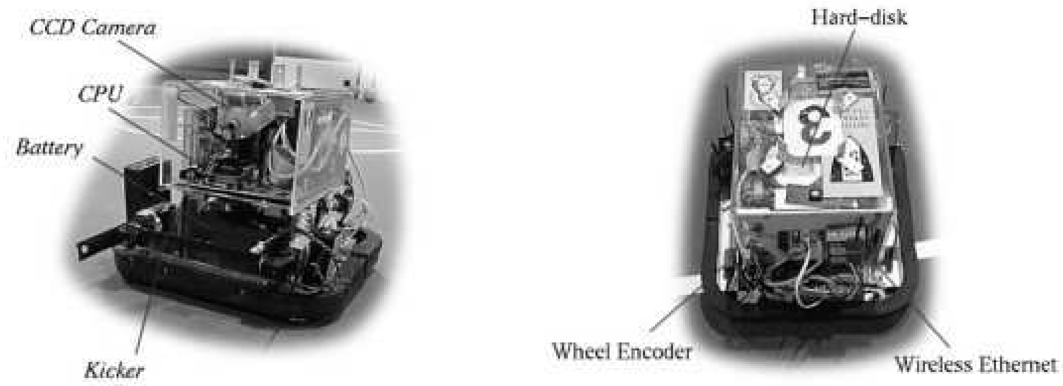


Figure 5.1: Individual Robot Hardware Architecture



a) Front View. b) Rear View.

Figure 5.2: Robot Platform

Figure 5.1 presents schematically the hardware functional architecture of the individual robots. Each mobile robot, besides having a processing unit, has vision capabilities to *see* the world, sensors to sense collisions and minimum distances and communication capabilities to interact directly with other robots. A module for absolute localization in known environments was considered and is currently under development. That module will use the field natural landmarks, such as the goals and the corners to determine the robots localization [12].

Figure 5.2 a) and b) present front and rear photos of one of the built robots. In *a)* it it is possible to see the CPU casing, the CCD camera, the front battery compartment and the kicker. In *b)* the wireless Ethernet model and the wheel encoders are visible.

Besides the hardware architecture, other aspects have to be considered in the specification of the mobile robots. The maximum volume, the weight and the payload. The volume was set with RoboCup rules in mind.

In the sequel, the different modules presented in Figure 5.1 are detailed.

5.1.1 Processing Capabilities

The robots processing unit must be powerful enough to implement the different levels of competence. It should also be scalable in order to receive future improvements and upgrades, i.e., it should have the power to run high-level and demanding algorithms.

Presently, the most cost effective systems are the common personal computers (PC), which become more powerful as price goes down. As a consequence of that trend, cheap and powerful hardware and software is now available in many flavors. As such, it was decided to provide each of the robots with a PC, equipped with RAM for program execution, a disk drive for program and data storage, Ethernet adaptor for communications and software downloading and video adapter for program development. All these components are commercial, off-the-shelf components. The robots can be connected to a docking station, equipped with a monitor, mouse, keyboard and battery charger. To interface the PC with the motors and the sensors, special hardware was built (*motor controllers boards* and *sensor interface*). Those interface the PC through the ISA bus. Device drivers were developed to deal with the special boards.

5.1.2 Mobility Configuration

The design of the mobility structure is a fundamental choice in a mobile robot. Given past experience [4] [31], a differential drive mobility structure was chosen. When compared with other solutions such as Ackerman steering, Synchro drive, omni-directional drive and trycicle drive, this kinematic structure has some advantages [22], namely:

- Mechanical simplicity.
- Stability.
- Maneuverability.
- Improved Controllability.

Despite its advantages, this kinematic structure has a major disadvantage: it is very hard to move on a straight line. This comes from the fact that it is virtually impossible to implement two wheel velocity controllers with exactly the same characteristics.

With this kinematic configuration, the vehicle kinematics is given by:

$$v(t) = \frac{r}{2}(w_{left}(t) + w_{right}(t)) \tag{5.1}$$

$$\dot{\theta}(t) = \frac{r}{\ell}(w_{left}(t) - w_{right}(t)) \tag{5.2}$$

where $v(t)$ is the vehicle linear velocity (m/s) and $w_{left}(t)$ and $w_{right}(t)$ are the left and right wheels angular velocity (rad/s), r is the wheel radius and ℓ is the distance between the wheels.

The mechanical platform is a commercial product. It is made out of wood and, in its original form was equipped with motors. In order to cope with the project needs, it was improved with bumpers, optical encoders, power amplifiers and batteries. It also holds the DC-DC converter that regulates battery power for the PC. The structure was further improved with the necessary cabling necessary to support a motherboard and a number of sensors and actuators.

5.1.3 Vision Sensors

In order to model the world, the robots must be capable of seeing it. At this point, the best cost-effective means of doing this is computer vision [10]. The size and cost of vision hardware (video cameras and acquisition boards) has dropped drastically [11], which has enabled vision systems to be embedded in mobile robots.

Vision provides a general sensor, i.e., with the same hardware many 'virtual sensors' can be implemented [32], from object recognition to obstacle avoidance. With it, it is possible to look for other robots, find and follow objects (namely a ball), identify features in the environment (useful for localization) and others. In order to make full use of the color information present in the RoboCup environment, it was decided to use a color camera.

A very important aspect when working with vision systems is the minimum time between image acquisitions, i.e., the time it takes to acquire, store an image into the computer memory and processing it (also referred as the *refresh rate*). It was decided to use a PCI acquisition board with direct memory access (DMA) because that technology has one of the best quality/price ratios in the market. Details on the implementation of the vision routines are given in [67].

5.1.4 Non-Vision Sensors

The developed robots were equipped with sensors to detect collisions and the proximity of objects. Those interface the central system trough a special board installed in the PC or via a serial, RS232 line.

The sensors are connected to the same communication bus (i^2c) and are autonomous in the sense that they have a local microcontroller that makes the measurements and pre-processes the data. When the central system pools the sensors they send back the latest sensed values.

The data read from the collision detector sensor is used to stop the robot in case of a collision. If that occurs and the motion is not stopped, the motors could be damaged. The proximity sensor is used to prevent such occurrences as it should make a prior warning of danger. It is prepared to give notice of an obstacle in a range of 30cm (can be a wall or another robot) in the near vicinity. In that case, the robot control system should detect the obstacle in order to avoid collision.

5.1.5 Kick Mechanism

The kicking ability enables the soccer player to move the ball into places that otherwise would be inaccessible. This item is useful for all players. With it, a Forward player can get the opponent Goal-Keeper by surprise or pass the ball to a better positioned teammate. A Goal-Keeper can also use it to make the ball move out of the defense zone.

A good kicking mechanism has the following capabilities:

- It provides a strong impulse to the ball.
- It is fast responsive.

In addition, with an ideal kicker, it would be possible to kick the ball with a given force in a given direction, pre-determined by the player. Those capabilities would be very useful to perform passes to other players.

A strong impulse guarantees that the ball is given enough speed to move away from the player. A fast response is essential due to the fast, dynamic environment where the robots operate.

This device interfaces the processing unit through a RS232 serial line.

5.1.6 Communications

In order to communicate with each other, the robots were equipped with RF wireless Ethernet modems. Those allow the broadcast of information and guarantee a global coverage in the soccer domain. Additionally, if an external computer is equipped with such a modem, it can be used for status monitoring and software download. Various communication architectures can be implemented with wireless Ethernet modems, from point-to-point to broadcast communication. In a sense, each robot acts as a *node* in a computer network.

5.1.7 Power Supply - Autonomy

The power supply of the robots is based on batteries. It has two electric sub-systems: signal electronics circuits (digital circuitry) and power electronics (to feed the motors and other demanding actuators). These two independent circuits are powered from different power sources. This is necessary because, when operating, power circuits (e.g. DC motors) induce current spikes in the power lines. If not controlled, these spikes may cause digital electronics malfunctions. Note that an isolation between the signals flowing from the signal electronics to the power circuits must be done. Those are optically isolated to provide the necessary galvanic isolation.

A note on the weight, size and recharging of the batteries must also be done. As the robots are small, the volume of the batteries must be kept small. Therefore, it is advisable to reduce the number of batteries to its minimum. A trade of must be made between the batteries price, volume, weight and capacity [29].

Battery recharging [13] for a team of autonomous robots is a complex logistics task because it is necessary to control the recharging of a large set of batteries with a small set of chargers in order to have charged batteries whenever necessary.

5.2 Test Setup

This section describes a setup for test and development of soccer abilities for soccer mobile robots. The setup includes a *test station* and a *set of tests* to be done with the robots. With it, it is possible to acquire results regarding the performance of the robots, in order to make improvements . Another use may be the adjusting of control parameters in order to meet design specifications (e.g., response time, ball following accuracy).

The test program was designed in order to provide incremental results, i.e., the robot is adjusted in an incremental way throughout the proposed tests and when skilled in some given task, it is possible to use it in the following tests.

5.2.1 Test Station

The test station has two modules:

- the soccer field;
- the test ramp;

Soccer Field

The soccer field is approximately a 4.5mx9m rectangle. The floor is covered with a green carpet and the walls are white. One of the goals is blue and the other is yellow.

Figure 5.3 presents a view of the field, with RoboCup98 compliant dimensions.

Test Slope

A test slope is an apparatus that allows a ball to be launched in a given direction with constant speed. In the tests, it is used to guarantee that the results are obtained with the same conditions, i.e., the ball speed is constant over a set of trials and the ball moving direction is the same. The slope inclination is controlled by the user (the ball velocity is proportional to the slope inclination). A schematic view can be seen in Figure 5.4.

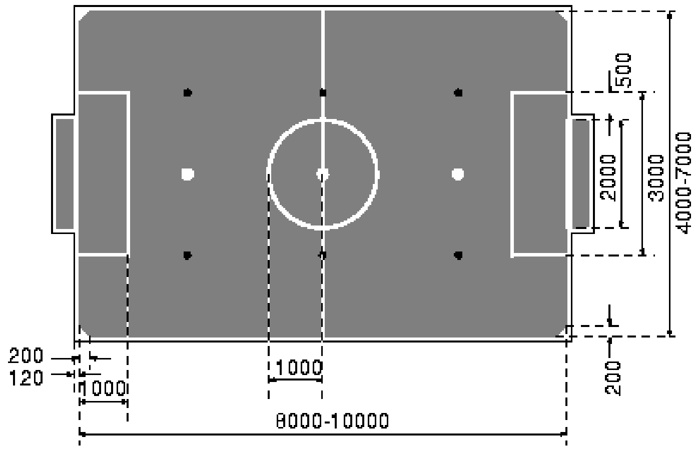


Figure 5.3: Soccer Field

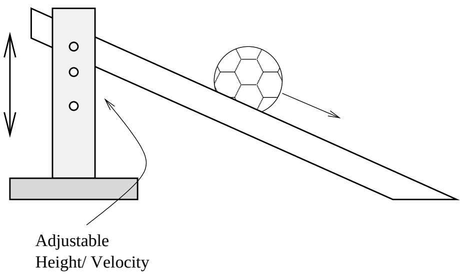


Figure 5.4: Side view of test slope apparatus

5.3 Tests and Results

The tests are inspired in Robocup Challenge [30]. Those tests are aimed at making a physical evaluation, exposing different aspects of the robots functionalities. Some examples of possible tests are presented in the sequel. The list is not complete and more tests can be added if necessary.

- Seeking a moving ball.
- Defending a goal.
- Intercepting and Controlling a moving ball.
- Moving a ball into a goal.
- Scoring penalties.
- Avoiding obstacles.

The tests results are registered in log files. The logged information may include information such as the elapsed time, the wheels speed set-points, the ball deviation from the center of the image and other parameters.

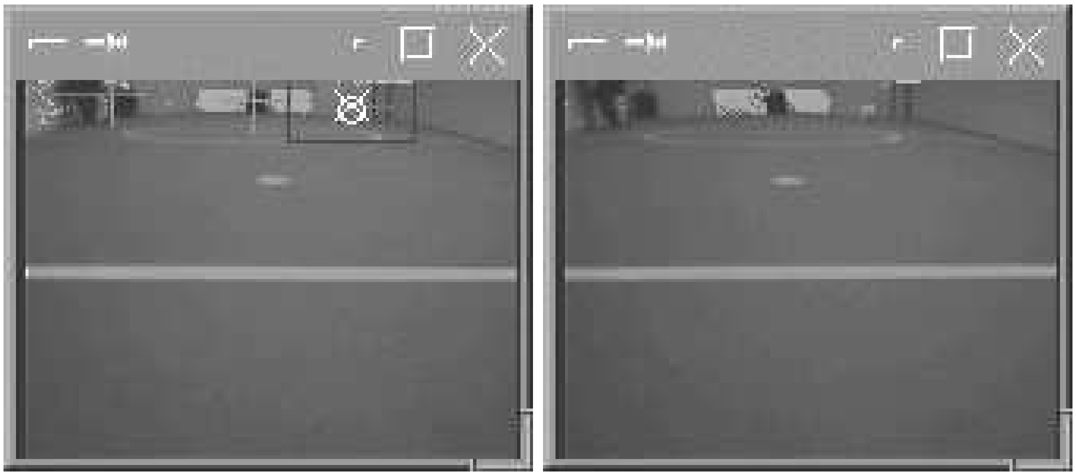
The same test can be done with different initial positions of the robot and of the ball launcher apparatus. The different configurations are assigned different codes in order to keep a precise record of the testings.

Robot Soccer Field

Figure 5.5 present real images, captured by the robots in the soccer field. Both images were taken from a robot placed in one of the goals, facing the other side of the field. In both images another robot is visible in the opponents goal, some robots in the top left area of the image and the ball in the top right corner of the image. In Figure 5.5 a) the capture parameters were correctly adjusted and the ball is marked with a cross. The opponents goal is also detected and its center of mass is also marked with a cross. The circle around the ball represents the ball radius estimate.

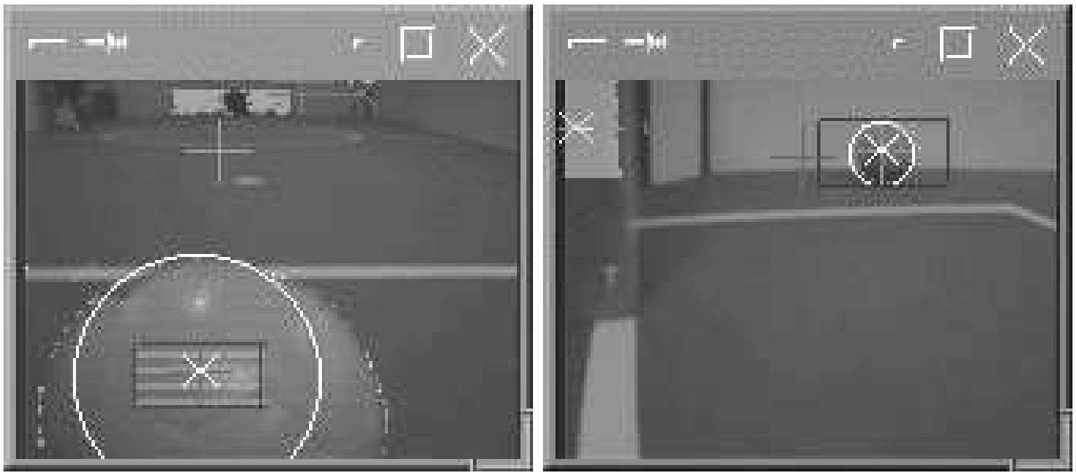
Figure 5.6 a) presents an image taken with the ball close to the robot. Figure 5.6 b) shows the image that the robot obtains when is looking to the left side of its goal (the goal interior wall is visible). A ball is placed in the vicinity and its radius is correctly estimated.

Figure 5.7 presents the evolution of the ball center of mass in the image when a ball was moved in the field, from the opponent goal vicinity into the team area. The robot was not moving while the ball moved.



a) Field Image. b) Field Image.

Figure 5.5: Real Images.



a) Ball Close to the Robot. b) Side View.

Figure 5.6: Ball Image and Side View.

Seeking the ball is what the goalkeeper does when it stays in the goal area while looking for the ball. In different tests, the ball is thrown from different positions, orientations and velocities. The robot is supposed to use only rotation to head towards the moving ball.

A schematic representation of the test setup can be seen if Figure 5.8. The objective is to move the ball in the field with different directions and velocities while a robot, placed in the G area tries to follow it using the $LOOK$ behavior. The ball trajectory presented in Figure 5.8 is an example.

Figure 5.9 shows some results obtained during this test. The shown lines represent the evolution of the ball center of mass in the image as the ball moves in the field. In this tests the ball was moved all over the field. The top of the figure shows

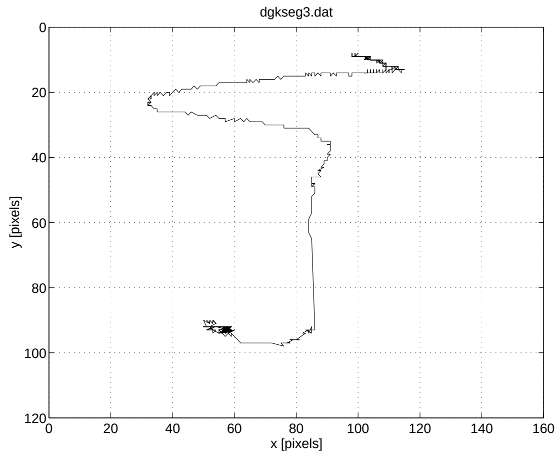


Figure 5.7: Ball Moving in the Field.

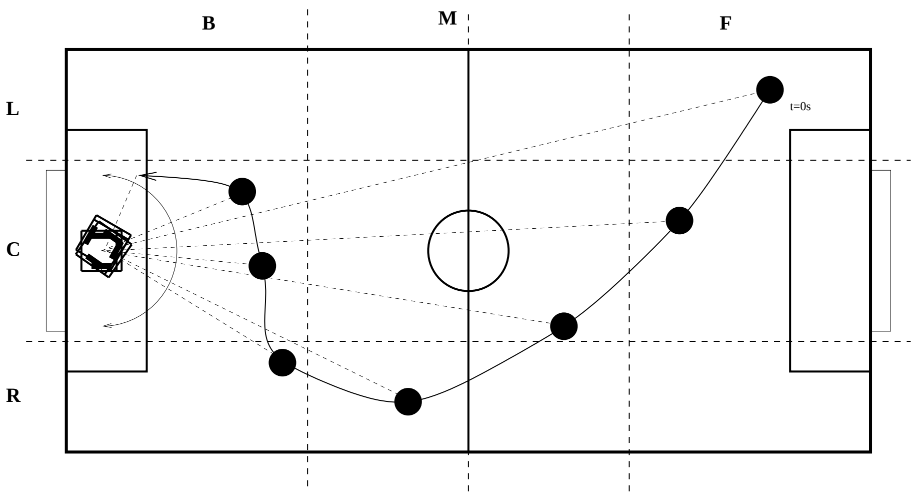


Figure 5.8: Ball Seeking test setup

larger density of lines due to the distortion caused by the camera position. In an ideal situation the ball center of mass in the image would allways stay close to the line `x=80` pixels. In Figure 5.6 a) and b) the half field division line is visible and is located in the upper quarter of the image. That mens that a player positioned in the goal area has low perception on the ball longitudinal motion in the opponents area (provided that there are no obstacles in the line of view) .

Figure 5.10 presents results obtained using two different hardware platforms for a similar test. In Figure 5.10 a) results obtained with the developed platform (SocRob) are presented. Figure 5.10 b) presents results obtained with a commercial platform (Scout – from Nomadic Technologies). The commercial robot computational platform is based on a personal computer (PC). An interface library was developed to interface its low-level control [19]. That library provides a compatibility layer

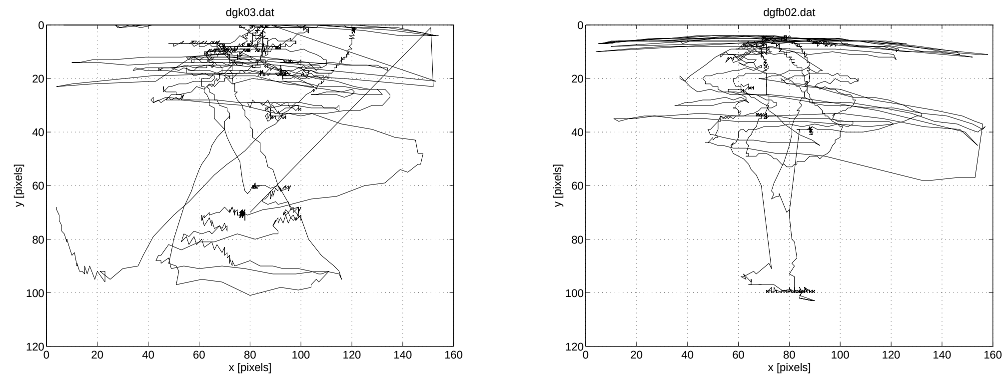
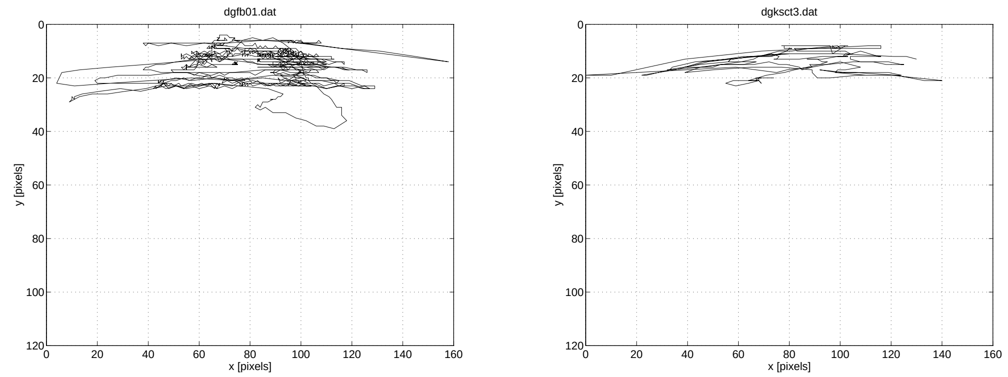


Figure 5.9: Seeking tests.



b) SocRob Robot. b) Scout Robot.

Figure 5.10: Seeking tests with different hardware platforms.

that makes the software compatible with both platforms (with minor parameter adjustments).

The ball center of mass in Figure 5.10 a) presents some "jumps" when compared with the results presented in Figure 5.10 b). Those differences are caused by the hardware. Scout robots have higher motion precision.

Defending a Goal Defending the goal is an essential capability for the soccer team. In this test, a player is assigned the Goalkeeper role and balls are kicked at the goal. For the set of launching positions and velocities, the measured parameters are the number of suffered goals.

Figure 5.11 presents the setup used to perform this test. The robot is assigned a goalkeeper role (LOOK-KICK, G). The ball is launched from different locations and with different directions and speeds. Results are presented in Table 5.1. Note that the tests were performed without obstacles. For each position, different velocities were used for the ball, by adjusting the slope inclination.

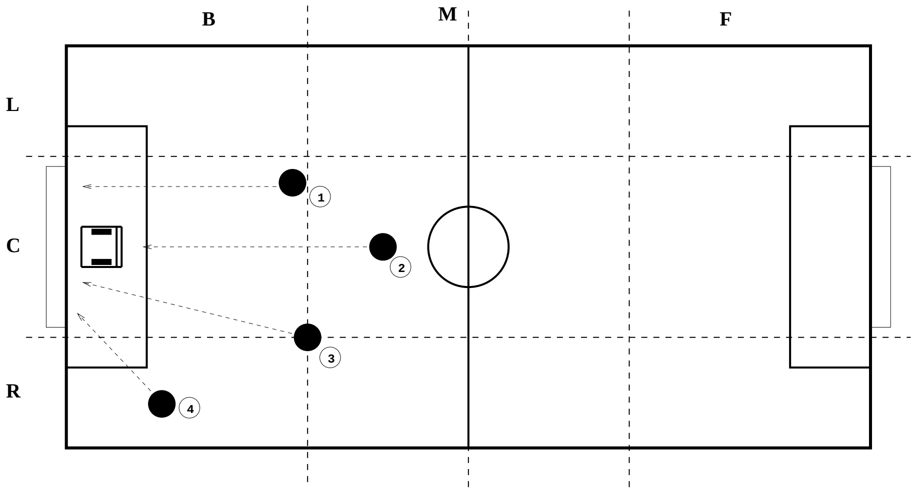


Figure 5.11: GoalKeeper test setup

Ball Start Position	*Suffered Goals*	*Saved Balls*
1	4	1
2	0	5
3	3	2
4	4	1

Table 5.1: Defending Results.

From the results presented in Table 5.1 it is possible to conclude that presently, this behavior is not good enough to implement a goalkeeper. It fails whenever the ball approaches the goal with a given angle (see Figure 5.11). This happens independently of the ball velocity.

Interception and Control of a moving ball is a import task for a soccer robotic player. This test aims at improving the player abilities to locate, follow and intercept and control a moving ball.

Placed in the G area, the robot is given the *FOLLOW-PUSH* behavior, that starts by looking for the ball, and, after locating the ball, starts moving towards it, keeping the ball in the center of the image, in order to take control over it. The ball is moved inside the field, in different directions and with different velocities.

Figure 5.12 presents some of the results obtained during this test. Almost no obstacles were present in the field. The two main line concentrations represent the ball close to the robot (bottom lines) and the ball far from the robot (top lines). The ball was sometimes moved away from the player in order to verify its reaction.

Moving to Goal Moving a ball into the goal area is essential to score goals. The

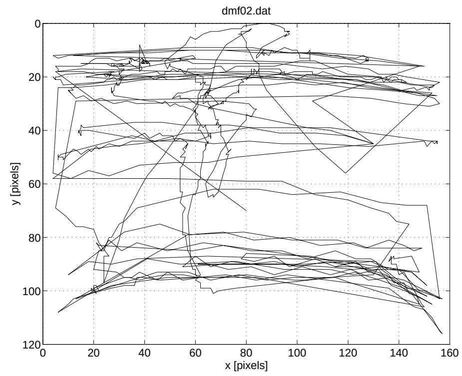


Figure 5.12: Intercept tests.

	Test 1	Test 2	Test 1	Test 2
Time	30s	56s	49s	107s

Table 5.2: Time to goal

objective in this test is to find the ball and move it into the goal, while controlling the ball, in order to be in a good position to shoot at the goal.

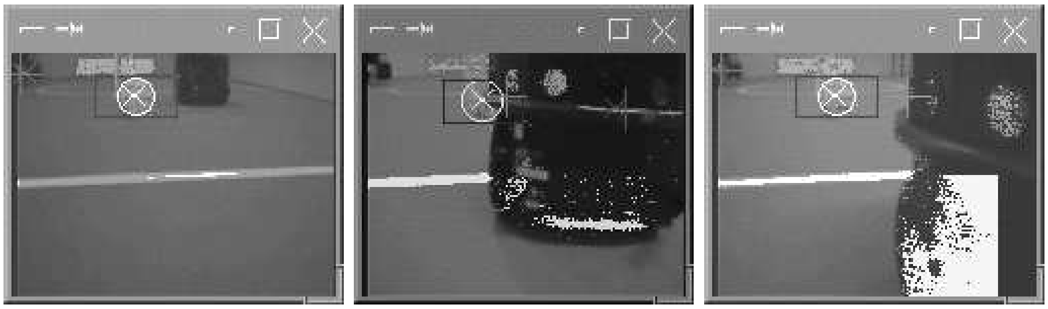
The test starts with the robot and the ball in random positions. It is given the *FOLLOW-KICK* behavior. The robot starts by looking for the ball, while tracing the last side it has seen the opponent goal. When the ball is found, the robot heads toward it. When close enough, it tries to position itself in order to have the ball and the opponent goal aligned. It that kicks the ball at the goal.

Table 5.2 presents some of the time stamps measured for a series of tests in different start positions. Note that the tests were performed without obstacles.

From Table 5.2 is is possible to conclude that the robot was able to move the ball, in a controlled form (most of the time), into the opponent goal area. However, the elapsed time has a close relation with the robot initial orientation with respect to the ball and the goal. This fact is strongly influenced by the fact that the robot is not able to self-locate inside the field.

Scoring penalties test is done with two robots. One of them is placed in the goal area. It is not logged. The second player has to move the ball into the goal, despite the other player opposition.

The test objective is to adjust the behavior in order to defeate the Goal-Keeper. Two players are used. To perform this test, one of the players is assigned the Goal-Keeper role and the other is assigned the Panality Shooter role. When a signal is given to the Penalty Shooter (through a remote keyboard), the penalty is started.



a) b) c)

Figure 5.13: Obstacle Avoidance.

With the current behaviors, no goals were scored. Although encouraging (from the Goal-Keeper stand point), this result may not hold when playing against other teams beacause the present Penalty shooter kicks the ball directy. A more efficient Penalty shooter could use information such as the distance between the Goal-Keeper and the side wall in order to kick at the zone less covered.

Avoiding obstacles is a constraint of the robot soccer game, since contact is not allowed. This test ainms at testing the robot ability to move in a hazardous environment, where it is surrounded by other robots or walls without colliding with them.

Figure 5.13 presents a sequence of real images taken in a situation where an opponent player is moved, manually, towards our goal. In image a) the ball and the opponent player are visible. The opponent player moves towards our goal. In b) the opponent player has already passed by the ball (that is still visible and is correctly located) and is entering the small area, close to our goal. The round shapes that are present in the top of the opponent robot are caused by reflexes of the walls in the robot sonars. In Figure 5.13 c) the opponent robot is near our robot. In the image the opponent presence is represented by a white area, that acts as a repulsor for our player.

Chapter 6

CONCLUSIONS

Despite not fully implemented yet, a multi-agent robotic system is now operational. It is composed of three autonomous mobile robots developed in this work framework.

The depicted functional architecture is being used to control the robots. The individual level is almost fully operational, except for the spatial supervisor as it depends on a sensor that is not available yet. A reference functional architecture was introduced and first steps towards its implementation have been taken. The envisaged data distribution and processing is fully implemented.

The developed infrastructure includes a population of mobile, autonomous robots with capabilities to do on-board processing, sensing, actuation and communications.

The target application to test the functional architecture was robotic soccer. Up to the present, the robotic team has participated in three scientific events: *Festival International des Sciences et Technologies* (1998), Robocup98 (1998) and Robocup99 (1999). In the first the team earned the 2nd place an the 1st place for industrial design. In Robocup98 the team was among the 8 quarter final competitors (out of 16 teams). In Robocup 99 the 9th position was attained (out of 21 teams).

Presently, the developed architecture and infrastructure has been used and further developed in the framework of the *SocRob* project. A graphical interface for the monitoring and control of the population was developed [19].

Furthermore, the architecture has already been used in another hardware platform (commercial robots) with a major success. The only changes were done in the library functions that access the motor controllers with some minor parameter adjustments.

6.1 Future Work

On-going efforts to extend the implementation of the reference functional architecture include:

- Development of the self-localization system. This item is essential to help robot guidance towards the ball and the goals, as well as to make them move to pre-defined locations and control their actuation areas. It is also part of the information regarding world state evaluation.
- Study and implementation of a teamwork model to be integrated in the functional architecture.
- Fine tuning the low level primitives, behaviors and roles in order to increase performance.
- Development of performance monitors in order to better understand the improvements, and choose among alternative primitive tasks, behaviors and roles.
- Implementation of the proposed architecture in a population working in another domain, such as rescue from catastrophe situations.

Bibliography

[1] R. Alamis, S. Fleury, M. Herrb, F. Ingrand, and F. Robert. Multi-robot Cooperation in the MARTHA Project. In *IEEE Robotics and Automation Magazine, Vol.5, No. 1*, March 1998.

[2] R. Alamis, F. Robert, F. Ingrand, and Shoji Suzuki. Multi-robot Cooperation through Incremental Plan-Merging. In *IEEE Robotics and Automation Magazine*, 1995.

[3] Alessandro Rubini. *Linux Device Drivers.* O'Reilly, Sebastopol, USA, 1998.

[4] Pedro Aparício, João Ferreira, Pedro Raposo, and Pedro Lima. Barbaneta - A Modular Autonomous Vehicle. In *Preprints of the 3rd IFAC Symposium on Intelligent Autonomous Vehicles*, March 1998.

[5] Pedro Aparício and Pedro Lima. Autonomous Distributed Control of a Population of Cooperative Robots. In *Proceedings of SIRS99*, September 1999.

[6] Ronald C. Arkin. *Behavior-Based Robotics.* MIT Press, 1998.

[7] Tucker Balch. *Behavioral Diversity in Learning Robot Teams.* PhD thesis, Georgia Institute of Technology, December 1998.

[8] Tucker Balch, Gary Boone, Tom Collins, Harold Forbes, Doug MacKenzie, and Juan Carlos Santamaría. Io, Ganymede and Callisto – a Multiagent Robot Trash-collecting Team. In *AI Magazine, Vol. 16, No 2*, Summer 1995.

[9] Kethryn Bartol and David Martin. *Management, 3rd Edition.* Mc Graw Hill, 1998.

[10] Tony Belpaeme and Andreas Birk. On the Watch. In *proceedings of ISATA97*, 1997.

[11] Tony Belpaeme and Andreas Birk. Working Toghether: Behavior-oriented Vision on Heterogeneous Autonomous Robots. In *Preprint submitted to Elsevier.* Elsevier Science, 1997.

[12] Margit Betke and Leonid Gurvis. Mobile Robot Localization using Landmarks.

[13] Andreas Birk. Autonomous Recharging of Mobile Robots. In *Proceedings of ISATA*, 1997.

[14] Andreas Birk and Tony Belpaeme. *A Multiagent System Based on Heterogeneous Robots*. Lecture Notes in AI 1456. Springer-Verlag, July 1998.

[15] Rodney A. Brooks. A Robust Layered Control System For A Mobile Robot. In *IEEE Transactions on Robotics And Automation, VolRA2, N1*, March 1986.

[16] Philippe Caloud, Wonyun Choi, Jean-Calude Latombe, Calude Le Pape, and Mark Yim. Indoor Automation With Many Mobile Robots. In *IEEE International Workshop on Intelligent Robots and Systems*, 1990.

[17] Y. Uny Cao, Alex S. Fukunaga, Andrew B. Kahng, and Frank Meng. Cooperative Mobile Robotics: Antecedents and Directions. In *Proc. of IEEE Int. Conf. on IROS*, 1995.

[18] Phil Cohen and Hector Levesque. Teamwork. Technical report, SRI International, March 1991.

[19] Bruno Correia, Sérgio Saraiva, Rodrigo Ventura, and Pedro Lima. Interface Gráfica para Operação de Robots Cooperantes. Technical Report RT-407-99, ISR, Instituto de Sistemas e Robótica, IST, PORTUGAL, September 1999.

[20] A. Drogoul and A. Collinot. Applying an Agent-Oriented Methodology to the Design of Artificial Organizations: A Case Study in Robotic Soccer. *Autonomous Agents and Multi-Agent Systems*, 1:113–129, 1998.

[21] J. Evans and B. Krishnamurthy. *HelpMate, The Trackless Robotic Courier*. Lecture Notes in Control and Information Sciences 236. Springer-Verlag, February 1998.

[22] H. R. Everett. *Sensors for Mobile Robots - Theory and Application*. AK Peters, 1995.

[23] Donald Franklin, Andrew Kahng, and Antony Lewis. Distributed Sensing and Probing with Multiple Search Agents: Toward System-level Landmine Detection Solutions. In *Proceedings of Detection Technologies for Mines and Minelike Targets, vol 2496*, 1995.

[24] Teruo Fujii, Hajime Asama, Takanori Fujita, Yuji Asakawa, Hayato Kaetsu, Akihiro Matsumoto, and Isao Endo. Knowledge Sharing Among Multiple Autonomous Mobile Robots Trough Indirect Communication Using Intelligent Data Carriers. In *Proceedings of IROS96*, 1996.

[25] Toshio Fukuda and Kosuke Sekiyama. *Cooperative Behavior Between Autonomous Agents.* Lecture Notes in Control and Information Sciences 236. Springer-Verlag, February 1998.

[26] Gregory Dudeck and Michael Jenkin and Evangelos Milios and David Wilkes. A Taxonomy for Multi-Agent Robotics. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1993.

[27] N.R. Jennings. Controlling Cooperative Problem Solving in Industrial Multi-Agent Systems Using Joint Intentions. In *IEEE Transactions on Robotics And Automation, VolRA2, N1*, March 1986.

[28] Jessica Perry Hekman. *Linux in a Nutshell.* O'Reilly, Sebastopol, USA, 1997.

[29] Joseph Jones and Anita Flynn. *Mobile Robots – Inspiration to Implementation.* AK Peters, 1993.

[30] Hiroaki Kitano, Minoru Asada, Itsuki Noda, and Hitoshi Matsubara. RoboCup: Robot World Cup. In *IEEE Robotics & Automation Magazine*, September 1998.

[31] Pedro Lima, M. Isabel Ribeiro, Pedro Aparício, and Renato Ferreira. Geometric Feasibility of a Flexible Cask Transportation System for ITER. In *Symposium on Fusion Technology*, March 1998.

[32] Pedro Lima, Paulo Silva, José Silva, and Carlos Cardeira. Centaur: A Vision-Based Autonomous Mobile Robot. In *European Advanced Robotics Systems Development (EUREL).* IEE Press, 1998.

[33] Pedro Lima, Rodrigo Ventura, Pedro Aparício, and Luís Custódio. A Functional Architecture for a Team of Fully Autonomous Cooperative Robots. In *International Joint Conference on Artifical Intelligence*, August 1999.

[34] Tim C. Lueth and Thomas Laengle. Task Description, Decomposition, and Allocation in a Distributed Autonomous Multi-Agent Robot System. In *IEEE/RSJ Conference on Intelligent Robots and Systems*, 1994.

[35] Sean Luke and Lee Spector. Envolving Teamwork and Coordination with Genetic Programming. In *proceedings on Genetic Programming*, 1996.

[36] R. Dillmann M. Kaiseer and O. Rogalla. Communication as the Basis for Learning in Multi-Agent Systems. In *ECAI-96 Workshop on Learnig in Distributed AI Systems*, 1996.

[37] Maja J. Mararić. Learning in Multi-Robot Systems. In *Eight European Conference on Machine Learning*, 199.

[38] Maja J. Mararić. Using Communication to Reduce Locality in Multi-Robot Learning. In *AAAI*, 1997.

[39] Maja J Matarić. Behaviour based systems. In *IEEE International Conference on Robotics and Automation*, 1992.

[40] Maja J Matarić. Integration of Representatin Into Goal-Driven Bahaviour Based Robots. In *IEEE Transactions on Robotics And Automation, Vol3, N3*, 1992.

[41] Maja J. Matarić. Minimizing Complexity in Controlling a Mobile Robot Population. In *IEEE International Conference on Robotics and Automation*, May 1992.

[42] Maja J. Matarić. *Interaction and Intelligent Behavior.* PhD thesis, Massachusetts Institute of Technology, May 1994.

[43] Maja J Matarić. Learning to Behave Socially. In *The Third International Conference on Simulatin of Adaptative Behavior*, 1994.

[44] Microchip. Microchip online. URL: `http://www.microchip.com`, 1999.

[45] Mirosot. Mirosot Contest and Exebition. URL: `http://www.mirosot.org`, 1999.

[46] Fabrice R. Noreils and Raja G. Chatila. Plan Execution Monitoring and Contrlo Architecture for Mobile Robots. In *IEEE Transactions on Robotics And Automation, Vol11, N2*, April 1995.

[47] Takua Ohko, Kazuo Hirako, and Yuichiro Anzai. Reducing Communication Load on Contract Net by Case-Based Reasoning – Eavesdropping for Utilizing Message Leakage. In *Procedings of IROS*, 1997.

[48] M. Muscholl P. Levi and TH. Braunl. Cooperative Mobile Robots Stuttgart: Architecture and Task. In *Intelligent Autonomous Systems*. IOS Press, 1995.

[49] Lynne E. Parker. Task-Oriented Multi-Robot learning in Behaviour-Based Systems. In *Procedings of IROS*, 1996.

[50] Suparerk Premvuit and Shin'ichi Yuta. Considerations on the Cooperation of Multiple Autonomous Mobile Robots. In *IEEE International Workshop on Intelligent Robots and Systems*, 1990.

[51] Ionanis Rekeleitis, Gregory Dudek, and Evangelos Milios. Multi-Robot Exploration of an Unknown Environment, Efficiently Reducing the Odometry Error. In *Proceedings of International Joint Conference in Artificial Intelligence, vol 2*. IEEE, 1997.

[52] M. Isabel Ribeiro, Pedro Lima, Pedro Aparício, and Renato Ferreira. Conceptual Study on Flexible Guidance and Navigation for ITER Remote Handling Transport Casks. In *Symposium on Fusion Engineering*, October 1997.

[53] RoboCup. Robocup online. URL: `http://www.robocup.org/`, 1998.

[54] RoboCup. Robocup rules. URL: `http://smart.informatik.uni-ulm.de/ /ROBOCUP/rules/`, 1998.

[55] Stuart Russell and Peter Norvig. *Artificial Intelligence, A Modern Approach*. Artificial Intelligence. Prentice-Hall, 1995.

[56] F. Sandt and L. Pampagnin. Perception for a Transport Robot in Public Environment. In *Proc.of IROS*, 1997.

[57] George N. Saridis. Intelligent Robotic Control. In *IEEE Transaction on Automatic Control, Vol. 28, N05*, May 1983.

[58] Miguel Schneider-Fontán and Maja J Matarić. Territorial Multi-Robot Task Division. In *IEEE Transactions on Robotics And Automation*, 1993.

[59] Peter Stone. *Layered Learning in Multi-Agent Systems*. PhD thesis, Carnegie Mellon University, December 1998.

[60] Peter Stone and Manuela Veloso. Multiagent Systems: A Survey from a Machine Learning Perspective. Technical Report CMU-CS-97-193, CMU, School of Computer Science, Carnegie Mellon University, May 1997.

[61] Milind Tambe. Teamwork in Real-World, Dynamic Environments. In *AAAI Fall Symposium on Plan Execution*, 1996.

[62] Milind Tambe. Towards Flexible Teamwork. In *Journal of Artificial Intelligence Research*, September 1997.

[63] Eiji Uchibe, Minoru Asada, and Koh Hosada. Behavior Coordination for a Mobile Robot Using Modular Reinforcement Learning. In *Proceedings of IROS*, 1996.

[64] Eji Uchibe, Minoru Asada, and Koh Hasada. Environment Complexity Control for a Visual-Based Learning Mobile Robot. In *proceedings of ICRA98*, 1998.

[65] Pascal Vasseur, Claude Pegard, and Laurent Delahoche. A Network for a Team of Cooperative Mobile Robots. In *Proceedings of Symposium on Robotics and Cybernetics*. IEEE-SMC, 1996.

[66] Rodrigo Ventura, Pedro Aparício, and Pedro Lima. Agent-based Programming Language for Multi-Agent Teams. Technical Report RT-401-99, ISR, Instituto de Sistemas e Robótica, IST, PORTUGAL, March 1999.

[67] Rodrigo Ventura, Pedro Aparício, and Pedro Lima. RoboCup98 ISocRob Team Technical Reference. Technical Report RT-702-99, RT-404-99, ISR, Instituto de Sistemas e Robótica, IST, PORTUGAL, July 1999.

[68] Rodrigo Ventura, Pedro Aparício, Pedro Lima, and Luís Custódio. Intelligent Society of Robots. In *Collection of RoboCup99 Team Description Papers*, August 1999.

[69] Rodrigo Ventura, Pedro Aparício, Pedro Lima, and Luís Custódio. *ISocRob – Intelligent Society of Robots*. RoboCup 99: Robot Soccer World Cup III. Springer-Verlag, 1999.

[70] Rodrigo Ventura, Pedro Aparício, Pedro Lima, and Carlos Pinto-Ferreira. SocRob — A Society of Cooperative Mobile Robots. In *Proceedings of 1998 IEEE International Conference on Systems, Man, and Cybernetics*. IEEE, 1998.

[71] Rodrigo Ventura, Pedro Aparício, Carlos Marques, Pedro Lima, and Carlos Pinto-Ferreira. *ISocRob – Team Description*. RoboCup 98: Robot Soccer World Cup II. Springer-Verlag, 1999.

[72] Rodrigo Ventura, Nuno Gonçalves, and Miguel Garção. A Programação por Agentes. Technical report, ISR, Instituto de Sistemas e Robótica, IST, PORTUGAL, July 1996.

[73] K. Yokota, K. Ozaki, N. Watanabe, D. Koyama A. Matsumoto, T. Ishikawa, K. Kawabata, H. Kaetsu, and H. Asama. Cooperative Team Play Based on Communication. In *Proceedings of the Second RoboCup Workshop*, July 1998.

Appendix A

SocRob Hardware

A.1 Design Issues

The robots of the population were developed to be integrated in scientific projects, whose requirements change over time. Its development was guided by design principles that are common to other areas [16] and capitalized past experience in the design of mobile robots [4] [32] and design of industrial prototypes [52] [31]. The main guidelines follow:

- *Modularity* is important to ensure that the platforms are reconfigurable and that the design is simple. With modularity we also mean that, should a component malfunction, it will be easily exchanged. Precious time is saved if some caution is taken at design time. It also means the development of different modules is made easier because a well defined interface exists.

- *Performance* concerns not only robot motion but also the reliability and robustness. The hardware must deliver adequate response, such as good maneuverability, extended operation time and efficient communications. In adition, the robot hardware and software should be reliable and robust to possible malfunctions.

- *Low Cost* is envisaged in all engineering projects. This one is no exception and all decisions were made with cost as a factor.

- *Off-the-shelf* and consumer electronics products were used whenever possible. In addition to ease of development, consumer electronics products are usually robust and power optimized.

The robots hardware (see Figure A.1) can be divided in five main blocks:

- Processing Unit.

- Sensorial System.
- Actuators.
- Communication System.
- Power.

Those are detailed in the sequel.

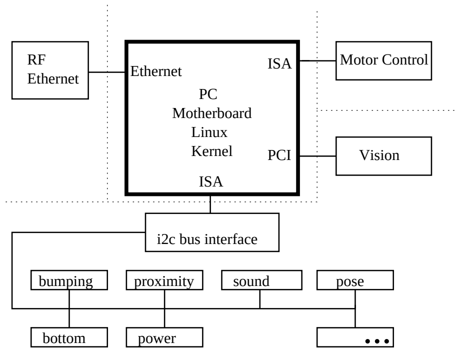


Figure A.1: Hardware Architecture.

A.2 Processing Unit

The robots main processing unit is based on a Personal Computer (PC). It includes the mother-board, the communications board (Ethernet), video board to plug a monitor for debugging purposes, a frame grabber board, mouse and keyboard. Additional boards include the motor controllers board and a i^2c bus interface board. Both motor and i^2c card interface the ISA bus on the motherboard. The video frame grabber interfaces through a PCI slot. The processor is an AMD K6, running at 200MHz. It has 16Mb available RAM memory and a 1.6Gb hard drive.

If necessary the hard-drive can be shutdown to increase mechanical robustness, data integrity and longer operation time.

A.3 Sensorial System

The sensors are divided in two main groups:

- **Vision Sensors**: These are virtual sensors that extract information from the video camera. One physical sensor leads to many virtual sensors.

- **Touch, Sound and Proximity Sensors**: each of those sensors is associated to a physical transducer.

All sensors but the video camera are connected to the central processing unit through a serial data bus. A PIC micro-controller [44] interfaces the sensors with the serial bus and pre-processes some of the data in advance.

This choice was made based on modularity, simplicity and robustness considerations. Each sensor has an unique identification number and was developed independently of the others.

The conceptual details of the sensors are presented in the sequel.

Video Camera — The video camera is a Phillips XC731/340 color camera, designed for home applications. It interfaces the PC system through a PCI captivator board. The combination allows the capture of 512×582 images at a frame rate of 50 interlaced images per second. It is placed 28,5cm above the floor, facing front and tilted 14° down, relative to the horizon. The acquired image is used for several purposes, namely seeking the ball and other robots, in the soccer domain. A auto-localization module, based on vision information is presently being developed.

Touch Sensors — The touch sensors are the last resource to a mobile robot about to collide with something or someone. Each sensor is composed of two switches, connected in series. The interrupters are linked by a plastic strip, to enlarge the sensing area.

Sound Sensor — It is not implemented yet. Would allow direct (voice or whistle) communication with the robots.

Proximity Sensor — The proximity sensors are based on Infra-Red (IR) technology. These sensors measure distances from 30 centimeters up to 1.5 meters. The proximity sensors can be used to implement an active collision avoidance procedure. Each robot has five proximity sensors.

Both *pose* and *distance* sensors use the same IR technology. The used detectors are comercial modules (GP1U58X - Sharp) that detect an IR modulated signal (40kHz). Internal filters make it immune from external noise such as visible light sources and magnetic fields.

A.4 Actuators

A.4.1 Mobility

The robots have a differential drive kinematic structure, i.e., they have individually controlled wheels. Additional caster wheels are provided for stability.

The velocity control is made in closed loop. A commercial velocity controller (LM629 - National) and optical encoders were used (500 ppr - HP). The velocity controller receives the velocity set-points from the PC, through the ISA bus, and closes the loop with the information acquired by optical encoders installed in contact with the wheels. PID control is used in this chip and it also allows for the seting of a aceleration ramp rate. The velocity controller output is pulse-width modulated (PWM) and feed to the power amplifiers. Those are based on the LM18200 (National) commercial H-bridge.

The connection between the velocity controller and the power amplifier is optically isolated in order to prevent malfunctions originated by power spikes. The need for two separate power sources originated the need for two on-board batteries (one for electronics and the other for motor power).

A.4.2 Kick Mechanism

Functionally, the kicker works autonomously from the rest of the system. It uses a modulated IR beam to detect the ball proximity. If the beam is interrupted, the kicker is activated. The kicker low-level control is done by a PIC micro-controller that is controlled by the motherboard through a serial line, and enabled or disabled by software.

Kicker Hardware

From a hardware point of view, the kick mechanism is composed by three different sub-systems. Those are: The Local Processing Unit, the IR beam/detector, and the actuation mechanism. Their inter-connections are presented in Figure A.2.

The power and command line shown in Figure A.2 is fed from the PIC micro-controller interface board, connecting to the robot sensors bus.

The kicker mechanics is based on a automobile door opening servomechanism. When powered with a given polarity, it moves the piston in one direction and when polarized in the opposite way, it moves in the other direction. The piston course is approximately *3cm*. Power for the servomechanisms comes from the motor batery.

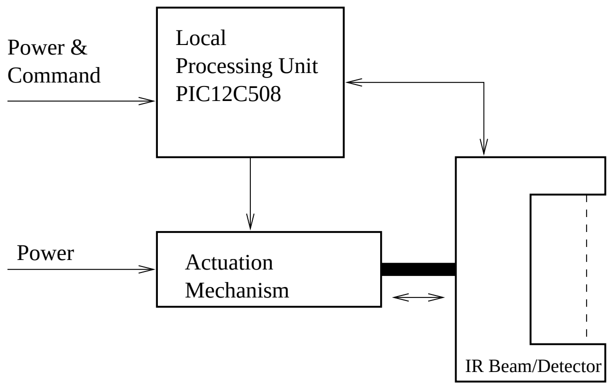


Figure A.2: Kicker Hardware Architecture.

A.5 Communications

To establish the communication channel, *RF Ethernet* modems were chosen (WaveCell from Aaron Tech). These allow communication in one of two frequencies: 2.4GHz e 2.4835GHz. The maximum available bandwidth is up to 2GHz and the modules can communicate to distances up to 150 meters. To ensure that the transmission medium is correctly shared, the modems use the CSMA/CA scheme (carrier sense multiple access with collision avoidance).

Figure A.3: Ethernet Modem.

A.6 Power

Power is a scarce resource in a mobile robot. It has to feed all internal modules, from the motors to the central processing and to the sensors. Past experience has taught that, if the system includes digital circuits close to power circuits, the power circuits *must* have a different ground. This comes from the fact that high consumption, inductive elements such as DC motors surge high currents in some situations, causing power failures. Without a ground separation, those occurrences would lead the digital electronic circuits (such as the motherboard) to malfunction.

The SocRob robots have two independent power sources, one for the digital electronics and sensors circuits and another for the motors and other demanding actuators.

The electronics is interfaced with the 12V7Ah battery through a DC-DC converter that transforms the battery 12V into the required +12, +5, -5 and -12 volts. The converter power is 45W.

Appendix B

Software

This appendix presents the robots operating system, a note on the enclosed device drivers, the information that is aquired from raw images, and some of the developed system functions.

B.1 Operative System

Linux is a freely available clone of the UNIX operating system for personal computers [28]. It was the choosed operative system for the robots. A careful analysis focused on the following issues:

- *Reliability* – Autonomous robots software must be robust. Therefore, not only the developed code must be robust but also the underlying operative system must be robust.
- *Communications support* – The developed robots must be able to comunicate with each other. The operative system should have embedded TCP/IP support.
- *Multi-tasking* – Multi-tasking is a valuable resource if one wants to use abstract functional divisions in the robots control. Robots interface with many i/o devices simultaneously.
- *Real Time* – Possibility of real time execution - Some robot tasks demand real time responses. Those must be handled by a real-time system.
- *Device drivers* – Robots hardware is controlled through device drivers. Those should not be overhelmingly dificult to write and debug.

Linux fullfils all the above listed requirements and in addition it has some extra advantages, namely:

- Tools - Development tools exist for a large set of languages such as (C, C++, assembly, lisp, etc).
- The price is low.

B.2 Device Drivers

A device driver is a software layer that lies between the application and the actual device [3]. It proveides the *mechanisms* to interact with hardware. In the developed robots, two non standard device drivers were used:

- *LM629 driver* – was developed from scratch. Makes the interface with the motor controllers, implementing almost all functionalities of the LM629 motor controller chip. This driver was developed by Rodrigo Ventura.
- *BTTV848 driver* – was *downloaded* from the `Internet`. Handles the access and control of the frame grabber board.

B.3 Vision - Acquired Information

The vision module presented in Figure 4.8 is responsible for the processing of raw images aquired by the camera. Being the primary source of information about the world, it must be reliable, robust, acurate and fast. Each image is processed in order to determine the center of mass and the area of the following objects in the image (details on color segmentation can be found in [67]):

- Ball.
- Opponents goal.
- Own goal.
- Obstacles in near vicinity.
- Near walls.

Given that the camera is fixed in the robot frame, it is possible to use that information to estimate some parameters, such as:

- the distance to the ball and other objects;
- the player location in the field;
- using a sequence of images, the ball velocity and its position in the future;

B.4 System Functions - Overview

System functions are the building blocks on which *primitive tasks* are built. Besides mapping the hardware and the software into robot abilities, these functions process the raw data and update internal variables that are later used by the above levels.

This functions are the result of a careful specification work. They are intended to cover the user basic needs when designing primitive tasks.

System functions are divided in several main areas: *Action*, *Sensing*, *Communication* and *Misc*.

Examples of such funcions follow:

motSetVel – Sets the velocity set-points. Accepts two parameters (v_{left}, v_{right});

motReset – Resets the motor controllers. Stops movement.

readContact – Reads the contact information;

getBall – Determines the coordinates of the center of mass of a ball present in the aquired image.

findCM – Given an image and a color, the center of mass and the mass of the objects in the image with that color is determined. It may be used for goal and ball determination.

findWhite – Determines the center of mass and the mass of the white color in the image.

findBlack – Determines the center of mass and the mass of the black color in the image. In the soccer domain the players are painted in black color.

setKick – Sets the kicker mechanism.

probeIsSocrob – Verifies if the hardware platform is a *SocRob* robot or a *Scout* robot.