Autonomous Distributed Control of a Population of Cooperative Robots*

Pedro Aparício and Pedro Lima**

Instituto de Sistemas e Robótica / Instituto Superior Técnico
Av. Rovisco Pais, 1, Torre Norte
1096-001 Lisboa
PORTUGAL

Abstract. Current and planned work on autonomous (i.e., with power supply and all computational power on-board) distributed control of a population of communicating cooperative robots is presented. A case study on Robotic soccer illustrates the main concepts involved.

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 [4] [12] [11] 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 [5] and RoboCup [8]). 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 [7], to transport food and medicines inside hospitals [9], are an advantageous solution regarding price and robustness, with respect to a one-robot system.

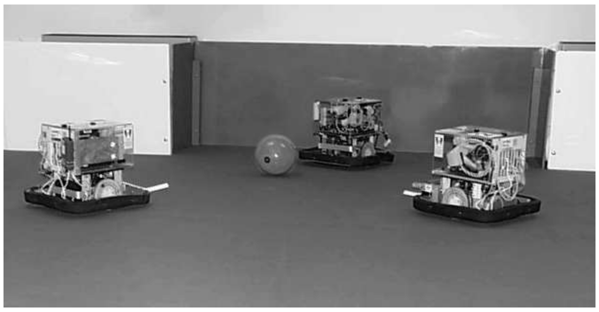


Fig. 1.: Developed Robot population

Building Multi-Agent Robotic Systems (MASR) is a challenging task as many options are 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 syntheses and control of this type of populations. This area gathers knowledge from many fields (like distributed systems) and therefore progresses very quickly.

When compared with single-agent systems, MARS offer a number of advantages. According to Stone [10] and Arkin [3], those include:

* This work has been supported by the following Portuguese institutions: Fundação Calouste Gulbenkian, Fundação para a Ciência e a Tecnologia, PRAXIS XXI/BM/12937/97.
** Email: {aparicio,pal}@isr.ist.utl.pt

- 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 are *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, simplifying the programming and testing of the individual sub-systems.

Albeit the advantages, MARS have some drawbacks:

- Coordination between the agents is difficult;
- 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.

This paper presents the development of a population of autonomous mobile robots (see Figure 1), endowed to perform cooperative tasks. The population hardware and functional architectures were designed with cooperation and modularity in mind.

1.1 A Case Study on Robotic Soccer – RoboCup

A case study on Robotic Soccer was set up to test and demonstrate the validity of the developed population, including the participation in RoboCup contests. RoboCup is the Robotic Soccer Wold Cup, organized in an annual basis by the RoboCup International Federation [8]. It is a contest that fosters Artificial Intelligence and Robotics research by providing a standard problem whose solution requires research on the integration of a wide range of technologies by using suitable architectures and performance measures. The design of a team to participate in RoboCup includes autonomous agents design principles, multi-agent collaboration, real-time reasoning, and sensor-fusion, to name a few. The developed population (*ISocRob*) [2] is composed of three homogeneous robots (regarding hardware). The robots have a differential drive mobility configuration, a video camera, infra-red distance sensors and RF Ethernet modems for inter-robot communications.

Figure 2 presents a functional division of the soccer field in several regions. These are zones used to simplify the location of the robots inside the field, during the game.

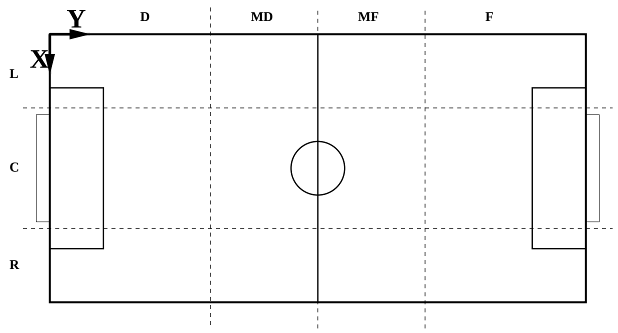


Fig. 2.: The field division in actuation areas.

Besides Defense (*D)*, MidDefense (*MD)*, MidForward (*MF)* and Forward (*F)*, further divisions are introduced to increase the field resolution. Along the field longitudinal axes, the field is divided in Left (*L*), Center (*C*) and Right (*R*) parts. Note that the goal area correspond to the *G* zone. Being used only by one player, it is not represented in Figure 2. The color scheme defined in the RoboCup rules indicates that the field floor is green, one goal is blue, the other is yellow, the robots are mostly black, the ball is orange and the walls are white. No global view of the field is allowed, and processing must be done on-board the robots.

The team is compliant with the RoboCup mid-size league rules and has been presented in *RoboCup98*, with a participation in *RoboCup99* envisaged.

1.2 Outline

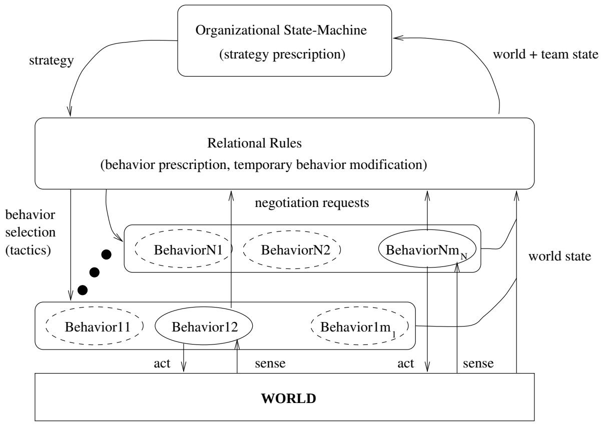
Section 2 goes through the population functional architecture. Its implementation details are presented in Section 3. Preliminary conclusions supported by currently achieved results are presented in Section 4.

2 Team Functional Architecture

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

The team architecture chosen to control the population is inspired on a 3-level functional hierarchy, first proposed by Drogoul and Collinot [6]. This architecture comprises three levels of competence. Our particularization for the control of a team of autonomous robots follows (see Figure 3):

- **Organizational**: This level establishes the strategy to be followed by the whole team, given the world state.
- **Relational**: At this level, groups of agents negotiate and eventually come to an agreement about some goal. At this level, based on the strategy prescribed by the organizational level, behaviors are distributed by the different agents, according to some tactics. Agreements are based on robots abilities, state and on the result of negotiation.
- **Individual**: This level encompasses all the available *behaviors*. Those include the *primitive tasks* (e.g., path planning, motion with collision avoidance) and their relations. All low level control issues are handled at this level.



Behavior ij : behavior i of agent j

Fig. 3.: Team Functional architecture

2.1 Organizational Level

The organizational level level is responsible for the overall control of the population. It decides on the *strategy* to be followed according to a given *world state* and application. In the soccer domain, the world state is composed of information regarding the players and the game state. The strategy comprises concepts such as `defend`, `counter-attack` and `attack`. Note that the different strategies are abstract concepts, i.e., no prescription on the individual behaviors of an agent is done at this level.

Although the organizational level is a central decision system, it differs from the common ones in the sense that it 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. In many applications, it is essential to have, at any time, an agent responsible for maintaining an overview of the world, in order to detect events which can be only determined from information aquired by the whole team. This is the robot where the organizational level is active.

2.2 Relational Level

The relational level is responsible for the translation of the orders sent from the organizational level (strategies to be followed) into executable actions (*tactics*). Tactics represents a mapping between strategies and team configuration pre-determined tactics (that correspond to pairs position/behavior in the soccer domain). This level runs simultaneously in all team members. It also gives the agents the capabilities to interact with others.

Given the strategy to follow (e.g., `defend` in the soccer domain), the active agents implement the prescribed *tactics*, using negotiation at relational level, to decide who should move where and what behavior to exhibit in that position. The possible conflicts between agents are presented and solved at this level. Note that, when doing task translations, a special care should be made in order to minimize changes in player behaviors and position. For instance, it would be a bad idea to have a goal-keeper and a forward player changing positions during a soccer game because it would open the goal for the duration of the change.

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 (probably, the best positioned). This kind of agreement has local scope (only the agents involved are aware of it) and it is stopped after a given time. Another example is a ball pass situation, where a player who has the ball asks one of its team mates to receive it.

2.3 Individual Level

In each agent, the individual level implements a *behavior* (and also a position in the soccer domain) that was prescribed at the relational level. Control issues related to sensor readings, data processing and actuation are handled at this level. Issues related with collision avoidance are also dealt with at this level.

Possible behaviors in the soccer robot concept include Goal-Keeper, Defender, MidField and Forward.

3 Implementation Issues

3.1 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. It also simplifies the process of adding agents to the team.

Besides the physical distribution, a conceptual distribution was also envisaged. The data is stored at different levels of abstraction, according to the control levels that use it. For example, in order to decide the strategy to be followed, the *organizational level* does not need to know the exact position of the players or if any of them is colliding with another. Instead, it uses information such as *which team has ball control* and *current score*.

Figure 4 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 or usefulness. Note that 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 the *organizational level* information from scratch, i.e., the data stored in the stopped player would be lost.

Figure 5 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` processes need information from the *sensors* (e.g. image) and from the *vision* process (e.g., ball position in image) in order to work.

Note that all information gathered by the different processes is stored in a common repository, accessible by all processes, not represented in Figure 5. 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 aquire and process it. This mechanism aims at reducing the computation time.

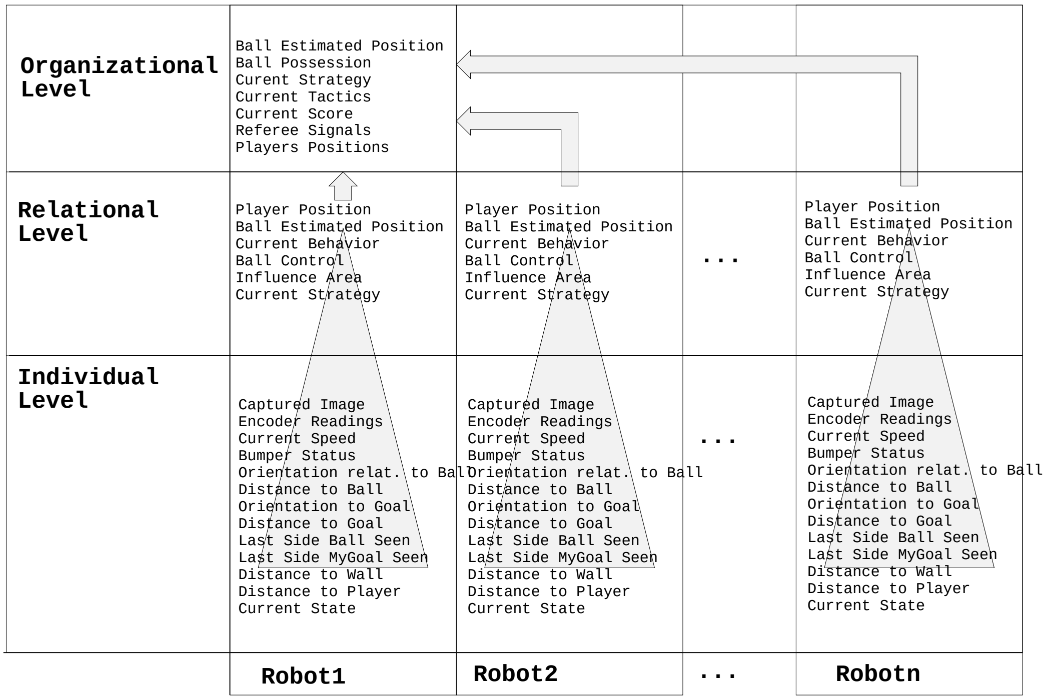


Fig. 4.: Sensorial data flow through the system.

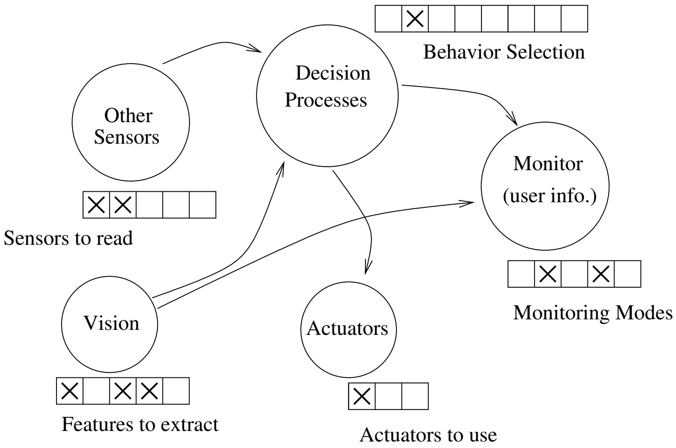


Fig. 5.: Processes and data flow.

3.2 Organizational Level

This state is implemented as a *state machine* whose states correspond to strategies sent to the relational level and whose transitions are traversed by changes in the world state.

The information stored in the world state at the *organizational level* results from the pre-processing of raw data (see Figure 4). From the organizational level point of view, its states can be segmented in two main categories: Game situations (e.g.,`kick-off`, `end-of-game`, `penalty-for`, `penalty-against`) and game evaluation (e.g., `ball-our-field`, `ball-our`, `elapsed-time`, `current-score`).

As a result of the two categories in the world state, the scenarios are divided in two main classes: *pre-programmed* scenarios for game situations and *dynamic* strategies such as `attack`, `defend` and `counter-attack`. As stated before, the transitions between states are traversed by changes in the world state.

Figure 6 presents a partial, schematic view of the organizational level implementation. The dynamic strategies are shown in the figure.

For the different world states, actions are prescribed to the relational level. Those are passed in the form

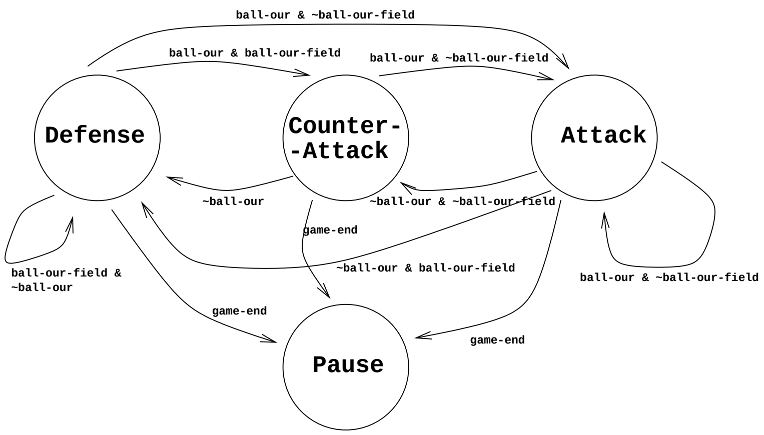


Fig. 6.: Organizational Level State Machine.

of a *strategy*, i.e., the organizational level passes to the relational level the recipe that should be applied to a given situation. For example, if our *team state* is `defense` and the *ball is in our field* and *one of our players has the ball*, the `counter-attack` strategy should be prescribed to the relational level. This prescription should lead the team to move up in the filed (move into the opponents field) and try to score a goal.

3.3 Relational Level

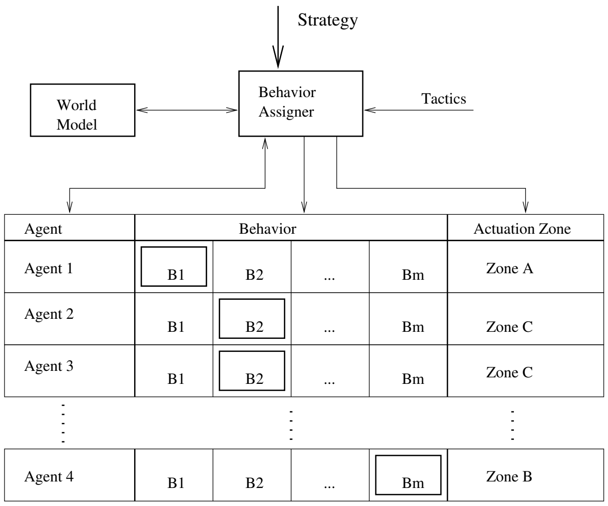


Fig. 7.: Relational Level – The highlighted boxes represent the active behavior in a agent.

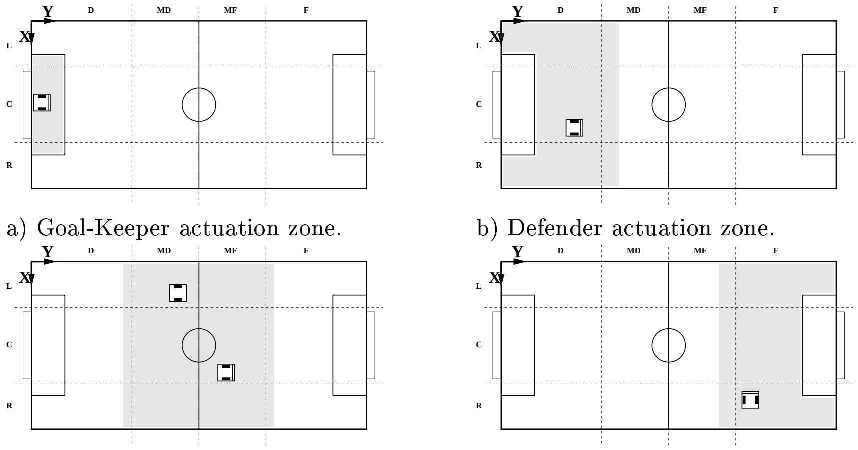
Figure 7 presents a schematic view of the relational level implementation. As stated before, when a strategy is received, it is converted into some (pre-defined) tactics that was a priory defined by the team manager. The tactics is stored at the relational level in each robot. When a different tactic is received, the relational level checks if its current behavior is required by the new tactics. If it is, it checks with the other relational levels (in the other robots) to look for possible incompatibilities (e.g., according to the mapping presented in Table 1, when moving from a defense strategy to a counter-attack, one of the defenders has to change to midfielder) and to solve them, if any. In order to disambiguate conflicts, simple rules are applied. If two robots have the same behavior but one has to change, the one that is nearest the new actuation zone is changed. If this criteria does not solve the problem, the time already spent in the behavior is used (the oldest behavior does not change) and, as a last resource, a random selection is made.

An example mapping presented in Table 1. It shows the zones and the behaviors that a number of players have to exhibit in order to implement a given tactic (assuming a team composed by four robots).

Tactics	Number of Players	Assigned Zone
Defense & Game Start	1	G
	2	D
	1	MD
Counter Attack	1	G
	1	D
	1	MD
	1	F
Attack	1	G
	1	MD
	1	MF
	1	F

Table 1.: Position mapping between Strategy and Tactics

The pre-defined possible zones are presented in Figure 8.



a) Goal-Keeper actuation zone.

b) Defender actuation zone.

c) Midfield actuation zone.

d) Forward actuation zone.

Fig. 8.: Actuation Zones

3.4 Individual Level

In order to implement the individual level of the functional architecture depicted in Section 2, the architecture shown in Figure 9 is implemented in each robot of the population. The *Behavior Coordinator* (top layer), selects the correct behavior for the robot, based on the *tactics* provided 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 field zones. The *spatial supervisor* ensures that the robot always stays inside its assigned area.

A *behavior* corresponds to a set of purposive (i.e., with a goal) states sequentially and/or concurrently executed. A *state* is composed by a *primitive task* and a list of 3-tuples. A 3-tuple consists of three components (logical condition; next state; priority). A *primitive task* consists of *sense-think-act loops* (STA loops).

STA loops 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:

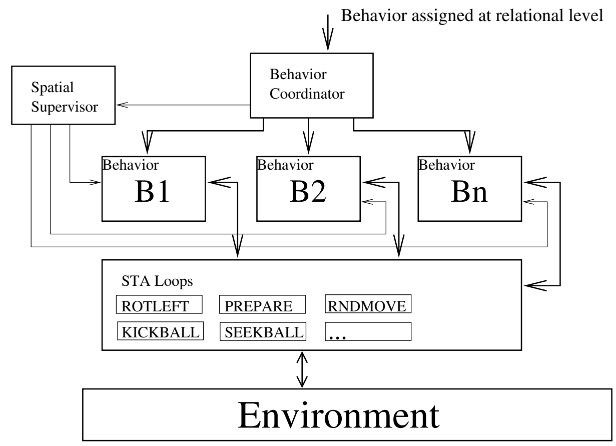


Fig. 9.: Individual Level.

- **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*.

The 3-tuples define the conditions to be met in order to make a state transition in the behavior execution. One or more conditions may exist on each state leading to different states. The priorities define a policy that ensures that the transitions are consistently transposed, without implementation dependencies. This concept is illustrated in Figure 10.

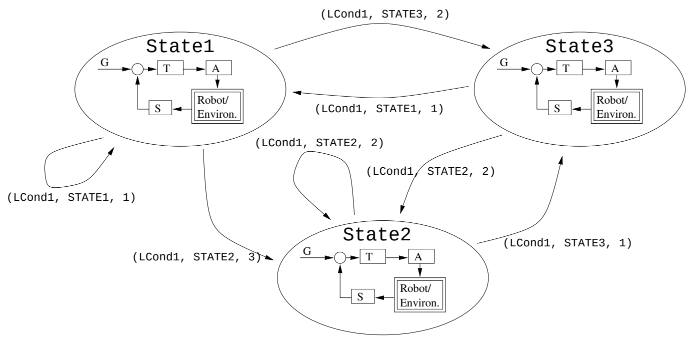


Fig. 10.: Behavior State Machine.

Each 3-tuple in the list is composed of the following key components:

logical condition – Logical conditions are 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`);

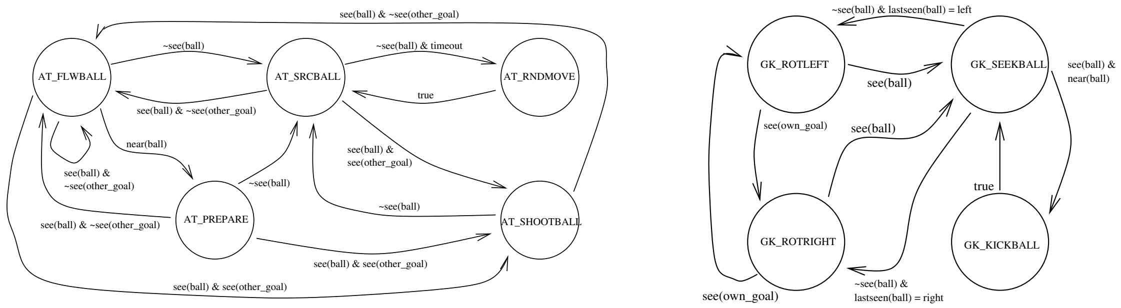
next state – The next state is the state reached upon the validation of the *logical condition* (e.g. `KICKBALL`);

priority – The priority disambiguates the *next state* to be reached if two *logical conditions* are true in the same evaluation (e.g., `1`). The behavior designer should not assign the same priority to any pair of 3-tuples. Nevertheless, if priorities are omitted, the next state will be choosen randomly;

Presently, the available *primitive tasks* to the design of behaviors comprise:

- `SrcBall` - orient the robot towards the ball;
- `SeekBall` - orinet the robot towards the ball and follow its movments, rotating around the robot axis;
- `FlwBall` - follow the ball;
- `RotDeg` - rotate `n` degrees;
- `ShootBall` - shoot the ball;
- `KickBall` - kick the ball and return to original position;
- `RndMove` - random move;
- `RotRight` - rotate continuously to the right;
- `RotLeft` - rotate continuously to the left;
- `Prepare` - manoeuvre in order to have the ball between the robot and the opponent goal.

At RoboCup98, two behaviors were implemented: Forward and GoalKeeper.



a) Forward behavior

b) Goal-Keeper behavior

Fig. 11.: Forward and Goal Keeper behavior in RoboCup98

The forward behavior, depicted in Figure 11-a), starts by searching the ball. When the ball is found, the robot moves towards it until a predefined distance is reached. Then, the forward players manoeuvres to get the ball between it and the goal. When that happens, the robot shoots the ball.

The Goal-Keeper behavior presented in Figure 11-b) is very simple. The robot looks for the ball, and if found, it follows it, rotating around its axis. If the ball approaches more than a given value, it tries to kick it away. If the ball is not visible, it starts rotating to the side the ball was seen for the last time. If the ball is not found but the own goal is, it starts rotating in the opposite direction. This simple behavior proved very efficient during the competition.

To better illustrate the STA loop and 3-tuple concepts, the the *primitive task* `GK_SEEKBALL`, present in the Goal-Keeper behavior (see Figure 11-b)) is detailed.

For this STA, the following key components are defined:

goal - Keep the ball in the center of the captured image (e.g., `XREF = 0`);
sense - Acquire and image to memory (e.g., `CaptureImage(img)`) and determine ball posiiton in image (e.g. `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));

where `XREF` represents the reference x coordinate in the image (e.g., central pixel), `xc` is the `x` coordinate of the ball in the image and `DSCALE` is a gain. `vc` and `vd` are the vehicle common and differential velocities.

The following 3-tuples are defined:

- (`see(ball) & near(ball)`, GK_KICKBALL,2);
- (`see(ball) & lastseen(ball) = left`, GK_ROTLEFT, 1);
- (`see(ball) & lastseen(ball) = right`, GK_ROTRIGHT, 1);

Note that equal priorities mean that the next state is choosen randomly.

4 Preliminary Conclusions and Future Work

Currently, our robots are capable of relatively simple behaviors (e.g. the Goal-Keeper above and a Forward simpler than the described), composed of primitive tasks, such as following a ball or shooting at the goal. These are sensor-based primitives, which means that collision avoidance (with walls and other robots) is included. Up to now, only vision has been used to implement these primitive tasks, but infra red and bump sensors are being installed around the robots to free image processing algorithms from obstacle avoidance requirements. A language based on the team and individual robot architectures is being developed to program the team at a high abstraction level, hiding low-level details from the programmer/strategist (e.g., the "coach", in robotic soccer). So far, the functional decomposition proposed in Section 2 is proving to be adequate to reduce the complexity involved in implementing all the required team capabilities, as it divides large problems in smaller sub-problems. This also paves the way for future work in team performance and evaluation, and performance-based decision-making when alternatives exist (e.g., different tactics - forward, mid-forward - to implement a given strategy).

Several behaviors will be implemented in the near future, such as `Defender` and `Forward`. New *primitive tasks* are also under development, in order to enable the development of new behaviors and the tuning of the current ones. Examples of under-development primitive tasks are: `Locate`, `PositionIn` and `DistanceTo`.

References

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. Pedro Aparício, Rodrigo Ventura, Pedro Lima, and Carlos Pinto-Ferreira. *ISocRob – Team Description*. RoboCup 98: Robot Soccer World Cup II. Springer-Verlag, 1999.
3. Ronald C. Arkin and Tucker Balch. Cooperative Multiagent Robotic Systems.
4. Tucker Balch. *Behavioral Diversity in Learning Robot Teams*. PhD thesis, Georgia Institute of Technology, December 1998.
5. 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*, January 1995.
6. 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.
7. J. Evans and B. Krishnamurthy. *HelpMate, The Trackless Robotic Courier*. Lecture Notes in Control and Information Sciences 236. Springer-Verlag, February 1998.
8. RoboCup. Robocup online. URL: `http://www.robocup.org/`, 1998.
9. F. Sandt and L. Pampagnin. Perception for a Transport Robot in Public Environment. In *Proc.of IROS*, 1997.
10. 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.
11. Peter Stone and Manuela Veloso. *Communications in domains with unreliable, single channel, low-bandwidth communications*. Lecture Notes in AI 1456. Springer-Verlag, July 1998.
12. Alex S. Fukunaga Y. Uny Cao, Andrew B. Kahng, and Frank Meng. Cooperative mobile robotics: Antecedents and directions. In *Proc. of IEEE Int. Conf. on IROS*, 1995.

This article was processed using the TEX macro package with SIRS99 style