S.P.Q.R.

D. Nardi, C. Castelpietra, A. Guidotti, M. Salerno, and C. Sanitati

Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113 - 00198 Roma, Italy

1 Introduction

The SPQR (Soccer Player Quadruped Robots, but also *Senatus PopulusQue Romanus*) team was a newcomer of the Sony Legged League in RoboCup 2000. The work started in April 2000, based on the previous experience gained in the Middle Size League [3] and on the collaboration of Artificial Intelligence and Robotics researchers of our Department. Due to the very short time to prepare the team for the competition, we decided to focus on the previously developed software architecture, based on the explicit representation of the knowledge of the robotic agent [1, 2] and on the effective realization of some control primitives, the kick in particular. Given the above constraints, the performance of the team in Melbourne was very satisfactory. In fact, SPQR classified fourth, winning games with more experienced teams, playing very tightly with the 99 winners in the semi-final, and generally showing a good level of play.

2 Team Development

Team Leader: Daniele Nardi (nardi@dis.uniroma1.it)
Team Members: Luigia Carlucci Aiello, Alessandro De Luca, Daniele Nardi* (Professors); Claudio Castelpietra *, Alice Guidotti *, Massimiliano Salerno*, Claudia Sanitati * (students).
Affiliation: same as above
Country: Italy
Web page: http://www.dis.uniroma1.it/~ leggedro

3 Architecture

In order to integrate deliberation and reactivity [4], we decided to build for our system a hybrid architecture with two layers [1]. In Figure 1 the functional modules forming our architecture are shown. A Perception module is involved in analyzing sensor data (especially from the camera and from the head and leg sensors): it stores and updates the internal world model. A Deliberation module receives the world model from Perception. It consists of a plan execution monitor, which controls the plan of actions to be executed, and of a set of primitive actions. These actions are then translated into commands either to our

* Attended to RoboCup 2000.

P. Stone, T. Balch, and G. Kraetzschmar (Eds.): RoboCup 2000, LNAI 2019, pp. 635-638, 2001.

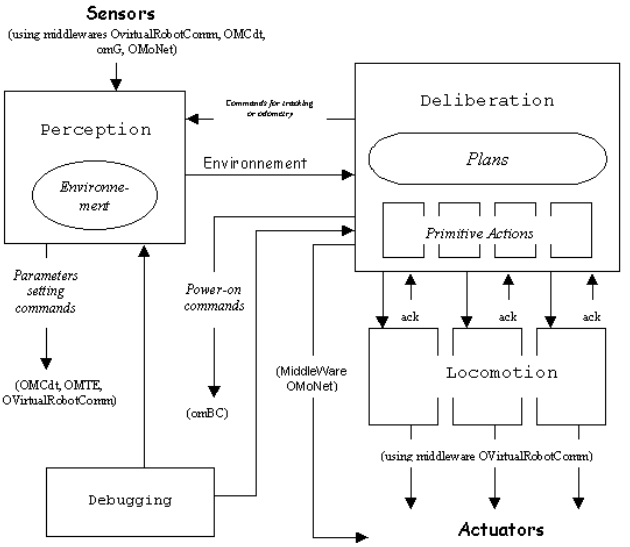

Fig. 1. Architecture

Locomotion module or to the OPEN-R middleware, OMoNet [5]. Locomotion is a module enclosing a set of four objects, each of which implements a different kick; this module provides an interface to the physical functionalities of the robot: it accepts abstract commands from the Deliberation module and translates them into effective commands to the actuators. A further module, named Debugging, is responsible for accepting debugging requests from the keyboard (through the serial connection to the robot): it allows printing sensor data on a terminal or storing images from the on-board camera in the memory stick.

The two-level layered architecture is based on a different representation of the information: there is a high-level symbolic representation, where knowing the exact position of the objects on the field is not necessary, and a low-level numerical representation, which instead is based upon the exact co-ordinates of the various objects. In the Deliberation module, there are: high-level plans, where the decisions on the actions to be executed are based on abstract conditions, such as *KnownBallPosition* or *NearBall*; primitive low-level atomic actions, that allow for an immediate reaction to unexpected situations and a certain amount of damping of the typical uncertainties of a dynamic environment.

4 Vision and Localization

Our vision system is based on AIBOs' hardware recognition and on a software module developed to build colour tables. The technique we used is to select, from a collection of pictures taken by the cameras of our robots, those areas containing the objects of the desired colour. We then check the Y, U, V value of

the pixels therein and set the extremes of the U and V ranges for each different value of Y, using an algorithm that aims at reducing, but cannot avoid, the intersections between colour tables.

Our localization is currently based on the recognition of two landmarks: the goals. This landmark recognition is done only when certain actions are being executed. Thus, there is no continuous localization and, at present, the robot does not determine its exact coordinates on the field. The objects' coordinates (ball and goals) are relative to the robots.

5 Behaviors

A Plan Execution Monitor is in charge of the correct execution of the actions composing the plans. In the monitor's implementation, a plan is stored as a graph data structure. The monitor's task is that of visiting the graph, calling and suspending the appropriate actions.

Each primitive implemented action is a C++ class derived from an abstract base class, named *pemAction*[2]. All the implemented actions have then basically the same structure. Each plan is an object of the *pemPlan* type, which is also a class derived from *pemAction*. This strategy allows plans and actions to be treated in the same manner, combining them in plans of higher hierarchical order. This approach has the advantage of being modular, extensible, readable and reusable. Moreover, as the plans are composed in growing hierarchical order, it is possible to design them leaving aside the robot physical platform or the implementation of the lower plans. In other words, each level uses the functionalities offered by the underlying levels, without caring about their implementation's details.

Furthermore, a plan selector allows the monitor to choose the current plan to be executed. In fact, besides the normal playing actions, the robot must deal with a set of particular situations, such as the initial kick or rising from a fall. These situations are managed by a set of specific plans, each of which has an associated condition. The monitor's task is that of verifying these conditions and of activating the higher priority plan whose activation condition is valid.

A library of boolean functions, based on the world model received from Perception, represents the abstract conditions that allow the state transition or the plan activation. These functions are based on the local co-ordinates of the ball and of the goals, and on the reliability of this information. A hysteresis mechanism allows the decisions to be stabilized [2].

We implemented more than thirty primitive actions: head, tail and leg movements are treated in separate primitive actions. For instance, a primitive action is in charge of all the head searching movement: the class constructor of this action takes, as a parameter, the type of search that is needed and, on the basis of this parameter the appropriate commands are send to the head. Another kind of atomic action manages all the commands that can be sent to the tail. More complex are the actions involved with the leg movements. They are based on

[2] pem stands for "plan execution monitor".

the local co-ordinates of the ball and goals. For instance, a *GoToBall* action calculates the angle between the ball and the robot direction and, depending on this angle, sends the appropriate movement commands to the robot: if the angle is small the robot must move straight forward, otherwise it must go to the right or to the left according to the angle's sign.

The primitive actions are combined together to form plans at different levels of abstraction. In a higher level plan, external conditions determine the plan to be activated next. We have three main high level plans, distinguished according to the roles of goalkeeper, defender (back) and forward.

Our robots have static co-ordination: this results from the differences between the plans for the various roles. In particular, the goalkeeper remains in front of its own goal and leaves it only if the ball is close enough, the defender tends to return to a backward position, while the forward attacks over the whole field. This strategy avoids interference among the robots and, most important, the execution of the same action at the same time, which would generally be counterproductive [2].

6 Action/Walking

We use the oMoNet service for walking, with a combination of default and fast walk, and kicking actions made by us. We developed two main kind of kicking actions: one using the front legs and the other using the head. In order to design the leg kick, we analyzed the cinematic of the leg and tried to maximize the speed in the motion direction. We then introduced this kicking action in a particular phase of the walking movement and we overlapped the kicking command to the walking command in order to make the action more effective through the help of the main-body push. The head kick can be frontal or lateral. Both these actions have been developed mainly by experimental tests.

7 Special Team Features and Conclusion

The main focus of our work has been the robots'architecture and the decision making system. In addition, we have designed several effective kicking actions.

In the future we intend to develop a more systematic approach to localization, to a robust vision system and to robot coordination.

References

1. Luca Iocchi. *Design and Development of Cognitive Robots.* PhD thesis, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", 1999.
2. D. Nardi. Artificial intelligence in robocup. In *Procs. of ECAI 2000.*
3. D. Nardi et al. ART-99: Azzurra Robot Team. In *Procs. of 3rd RoboCup Workshop.*
4. A. Saffiotti. Some notes on the integration of planning and reactivity in autonomous mobile robots. In *Procs. of the AAAI Spring Symposium on Foundations of Automatic Planning*, pages 122–126, Standford, CA, 1993.
5. Sony Corporation. *OPEN-R Software Programer's Guide*, 2000.