Software Agents for Autonomous Robots: the Eurobot 2006 Experience

Vincenzo Nicosia[1], Concetto Spampinato[1] and Corrado Santoro[2] for the Eurobot DIIT Team
Università di Catania
[1]Facoltà di Ingegneria - Dipartimento di Ingegneria Informatica e delle Telecomunicazioni
[2]Facoltà di Informatica - Dipartimento di Matematica e Informatica
Viale A. Doria, 6 - 95125, Catania, Italy

***Abstract*— Agent-based software architectures have been used and exploited in many application fields. In this paper, we report our experience about using intelligent agents for an unusual task: controlling an autonomous robot playing a kind of "golf" game in an international robotic competition. Driving a real robot is a practical application field for software agents, because different subsystems need to be controlled and synchronised in order to realize a global game strategy: cooperating agents can easily fit the target. Since this application requires a soft real-time platform to guarantee fast and reliable actions, and also a valuable communication system to gain feedback from sensors and to issue commands to actuators, we chose Erlang as programming language. A two-layer multi-agent system was thus designed and realized, composed of a lower layer, hosting agents taking care of the interface with sensors and actuators, and a higher layer, where agents are in charge of "intelligent" activities related to game strategy.**

***Keywords*—** Mobile and Autonomous Robots, Computer Vision, Autonomous Agents, Real-Time Systems, Erlang.

I. Introduction

Software agents are autonomous entities that, living in a virtual world, are in charge of accomplishing the goal they are programmed for. In doing so, agents interact with the environment where they live in, by sensing its state and acting onto it, in order to achieve their goal. For these reasons, they are often called "software robots".

In spite of this similarity between (software) agents and (real) robots, agents, and above all multi-agent systems, are mainly exploited in realizing complex software systems and applications requiring intelligence, flexibility, interoperability, etc., while the area of robotics is often a matter of research on real-time and control systems. However, when a (autonomous) robot needs some intelligence to perform its activities in a more efficient and effective manner, the use of agent technology seems a natural choice [17].

The issue is that, in these cases, agents have to face the problems related to the interface to physical sensors and actuators, which connect the computer system with a physical environment that also changes during time. Therefore, an agent–enabled robot has not only to tackle the problems related to direct use of input/output ports, acquisition and driving boards, serial ports etc., but it should also take in account the fact that the scenario is time-constrained. In fact, as it is known, an information acquired from sensors (e.g. the position of the robot or of its arm) has a deadline after which the data become stale and no more useful, unless a fresh value is obtained. These problems are quite known in the area of real-time systems and their solution is achieved by means of platforms and/or operating systems that regulate program execution—in terms of process/task scheduling, race condition and delay control—in order to guarantee that deadlines are met.

Since such a real-time support is needed also in the case of the use of an agent-based system to control robot activities, the traditional and well-known agent platforms, which are mainly based on Java, cannot be employed at all: at it is known, the main problem of Java is the garbage collector, which introduces unpredictable latencies that prevent any attempt to build a time-constrained system. Indeed, RTSJ specification [6] provides a set of classes and some programming rules that allow the realization of real-time Java systems, but the specification introduces hard constraints in object allocation and reference that require an existing Java program (and thus an agent platform) to be rewritten in order to make it RTSJ-compliant [22], [16], [8].

In the context of agents and real-time systems, a language that features some interesting characteristics is *Erlang* [5], [4], [1]. It is a functional and symbolic programming language that has been proved to be suitable for the implementation of multi-agent and intelligent systems [21], [10], [12], [11], [13], [15], [14], [9]; moreover, since the Erlang runtime system is able to provide *soft real-time*[1] capabilities [18], [3], it seems also quite useful for the realization of an autonomous robot controlled by autonomous agents. In this context, this paper describes the authors' experience in designing and implementing an autonomous robot, for the Eurobot 2006 competition[2,3], by means of a multi-agent system written using the Erlang programming language. A layered multi-agent system has been designed, composed of two layers: a *back-end* (lower layer), comprising agents performing the interface with robot's physical sensors and actuators, and handling low-level control activities; and a *front-end* (upper layer), hosting agents dealing with the game strategy. Thanks to this layered architecture, hardware-level interactions and

[1]A system is called *soft real-time* if it is able to take into account deadlines, but if a deadline is not met, it has no particular consequences [19], [20].

[2]`http://www.eurobot.org`

[3]`http://pciso.diit.unict.it/~eurobot`

intelligent activities are clearly decoupled, making the design and implementation of the software system more easy, and also allowing the programmer to easily reuse some parts and/or to improve or change the functionalities of the system.

The paper is structured as follows. Section II describes the game that robots have to play at Eurobot 2006. Section III illustrates the basic hardware and mechanical structure of the robot developed. Section IV deals with the software architecture of the control system of the robot, describing the agents composing the system, their role and their activities. Section V discusses some implementation issues. Section VI reports our conclusions.

II. The Game at Eurobot 2006

Eurobot is an international robotics competition which involves students and amateurs in challenging and amazing robot games. The main target of the event is to encourage sharing of technical knowledge and creativity among students and young people from Europe and, in the last two editions, from all around the world.

Every year a different robotic game is chosen, so that all teams start from the same initial status and new teams are stimulated to participate. Here we report an overview of the rules for the 2006 edition of Eurobot[4], when the selected game was "Funny Golf", a simplified version of a golf game where robots had to search balls in the play-field and to put them into holes of a predefined colour.

A. Field and Game Concepts

As Figure 1 shows, the play-field is a green rectangle of 210x300 mm, surrounded by a wooden border. Borders on the short sides of the field have a red (resp. blue) central stripe which delimits the starting area for each robot. The field has 28 holes, 14 of them encircled by red rings and the other by blue rings. A total amount of 31 white balls and 10 black balls are available during the game. Fifteen white balls and two black balls are placed into the playing area at predefined positions, while four more black balls are randomly positioned into holes, two for each colour. The remaining balls (sixteen white and four black) could be released by automatic ejection mechanisms positioned at each corner of the field. Finally, four yellow "totems" are positioned into the field and are both obstacles for robots and switches for the ball–ejection mechanisms.

Robots must be absolutely autonomous: any kind of communication with the robot, both wired or wireless, is not allowed during matches. Robots have spatial limits, in terms of height, perimeter and so on, and have to pass a homologation test before being accepted for the competition. Each robot can also use any kind of positioning and obstacle-avoidance system, and supports are provided at the borders of the playing area to place (homologated) beacons, if needed.

[4]This edition took place in Catania, Italy.

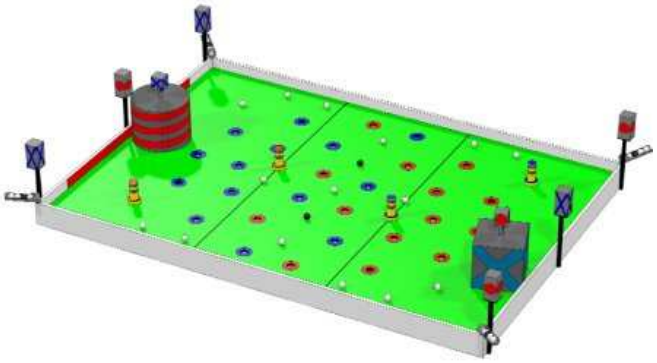


Fig. 1. The playing area

B. Playing Funny Golf

Before starting, each robot is assigned a colour, either red or blue. Robots start from the border opposite to their playing area, i.e. in the opponent's field, and at least one side of the robot must touch the starting area (short border of the play-field). After robots are placed into the field and all setup procedures by team members are over, the referees choose the positions of totems and black balls, by means of a random selection. When all the components in the play-field are set up, one of the referees gives the start signal and robots can play. Each robot has to put as many white balls as possible into its holes in a time of 90 seconds. Robots can also put black balls into opponent's holes, suck them out of their holes, or even suck white balls out of opponent's holes. There is no restriction about strategies or techniques adopted in order to search, catch, release and suck out balls. It is not allowed to hurt the other robot or to obstacle or damage it in any way. It is neither permitted to damage the playing area or playing objects (such as balls, holes, totems or ejecting mechanisms). The ejecting mechanisms can be triggered by touching a totem for a given amount of time: this closes a simple electric circuit and allow balls into the ejector to be released. At the end of the match, each white ball in the right hole is considered as a point, and the robots which has the highest score is the winner.

III. The DIIT Team Robot

Building an autonomous robot to play "Funny Golf" is not a trivial task, since different subsystems are needed to perform ball searching, catching and putting, and many physical constraints are imposed by game rules themselves. The following subsections describe the robot realized by the DIIT Team[5], which participates (for the first time) to the 2006 Eurobot edition.

A. The Core

An embedded VIA 900Mhz CPU is the core of the robot. We used a motherboard produced by AXIOM Inc. which incorporates Ethernet, parallel port, 4 serial ports, USB, IDE

[5]"DIIT" means Dipartimento di Ingegneria Informatica e delle Telecomunicazioni.

controller and other amenities (such as PC/104 bus, not used in our configuration). The operating system used is a Debian GNU/Linux (Etch), with kernel 2.6.12 and glibc 2.3.5. GNU/Linux was selected because of its stability and robustness, that are important features when driving a robot.

B. Locomotion System

In order to guarantee fast movements, we decided to use a locomotion system based on two independent double-wheels, driven by DC motors. Wheels diameter is small enough to allow fast rotation and large enough to avoid holes. DC motors are directly connected to a motor-controller, driven by a RS232 serial line. The controller allows to set different speeds for each wheel, both for forward and backward directions. Each wheel is connected to an optical encoder, driven by a serial mouse circuitry, which feeds back to the software system information about real rotation speed and position of the wheel. This information is then used by the Motion Control agent to adjust the speed and the trajectory.

C. Vision

Searching balls in the playing area requires a kind of vision system to find them. We chose to use a simple USB webcam to capture video frames at a rate of about 4 frames/sec, still enough to guarantee an accurate and fast analysis of objects in the field. The webcam is able to "view" the field from 30 to 160 centimetres in front of the robot, with a visual angle of about 100 degrees in total. Frame grabbed by the webcam are passed to the "Object Detector" agent, which filters them to find balls (both black and white) and holes (both red and blue).

D. Catching and putting balls

Once balls are detected, it is necessary to put them, somehow, into the right hole. We decided to suck balls using a fan, and to choose where to put them using a simple selector, driven by a servo-motor. Balls are saved into a small buffer if they are white and the buffer has enough space, or ejected out if they are black or if the buffer is full. The fan is powerful enough to suck balls at a distance of about 12 centimetres from the front side of the robot, and it is also able to suck balls out of holes when a special small bulkhead on the front side is closed. A simple release mechanism, which uses a servo-motor, allows balls to be dropped down to the final piece of the buffer and to fall into a hole.

E. Sensors and Positioning

Many sensors have been used onto the robot. First of all, a colour sensor for balls is installed into the ball selector, to recognise if a sucked ball is white or black. A complex system of proximity sensors is installed in the bottom side of the robot to recognise holes when the robot walks over them, and to allow a smart and fine positioning during the ball putting phase. A presence sensor (made by a simple LED–photo-resistor couple) is placed in the final part of the buffer, to reveal the presence of a ball ready to be dropped into a hole. The same sensor is used to detect when the ball has been successfully put into a hole.

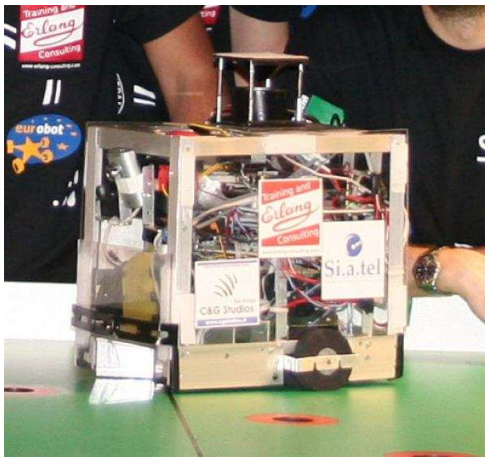


Fig. 2. The Robot in the playing area

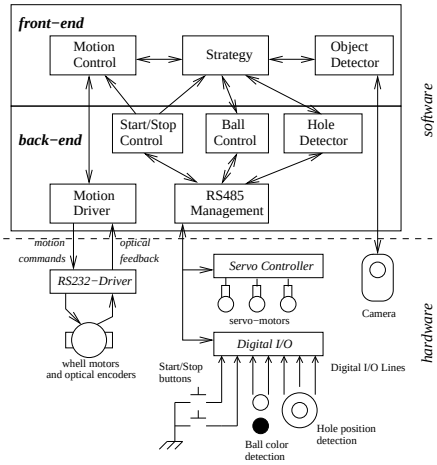


Fig. 3. Hardware/Software Architecture

IV. The Robot's Software Architecture

Given the robot structure illustrated in the previous Section, it is clear that the implementation of the system to control it has to face some problems that are not present in traditional (only software) multi-agent systems: the interface with physical sensors and actuators. For this reason, the basic software architecture of the robot, which is sketched in Figure 3, is composed of *two layers*, (*i*) a lower one, called the *back-end*, including reactive-only agents, responsible for a direct interaction with the hardware, and (*ii*) a higher layer, called the *front-end*, hosting the "robot's intelligence" by means of a set of agents implementing the artificial vision system, the game strategy, the motion control, etc., and interacting with back-end's agents in order to sense and act onto the environment. All of these agents comply with an ad-hoc model which, together with the details on functionality of the overall system, is described in the following Subsections.

A. Agent Model

As reported in Section I, due to real time requirements and other peculiarities of a robotic application, well-know Java-based agent platforms cannot be employed; therefore, according to authors' past research work [21], [10], [12], [11], [13], [15], [14], [9], we decided to use the Erlang language [5], [4], [1] for the development of the robot's software system. In addition to its soft-real time features, Erlang has a concurrent and distributed programming model that perfectly fit the model of multi-agent systems: an Erlang application is in fact composed by a set of *independent processes*, each having a *state*, *sharing nothing* with other processes and communicating only by means of *message passing*. Such processes can be all local (i.e. in the same PC) or spread over a computer network; this is transparent to the application because the language constructs for sending and receiving messages do not change should the interacting processes be local or remote.

Given these features and the requirements for the robot control application, a suited agent model has been developed, which is based on two abstractions called *BasicFSM* and *PeriodicFSM*. The former, *BasicFSM*, is essentially a finite-state machine model, in which transitions are triggered by either the arrival of a message or the elapsing of a given timeout, and a specified per-state activity is executed (one-shot) when a new state is reached. The latter, *PeriodicFSM*, is instead a finite-state machine in which transitions are activated only by the arrival of a message, while the per-state activity is executed, when a state is reached, *periodically*, according to a fixed time period and within a deadline, which is equal to the period itself.

As it will be illustrated in the following, BasicFSM model is used for front-end agents, while the PeriodicFSM model is essentially exploited for those interacting with sensors and actuators and thus running in the back-end.

B. The Back-End

As Figure 3 illustrates, the *back-end* layer is composed by the following agents: *Motion Driver*, *RS485 Management*, *Start/Stop Control*, *Ball Control* and *Hole Detector*. All of these agents use the PeriodicFSM model but only the first two are directly connected with hardware resources.

The *Motion Driver* agent is in charge of driving wheel motors and gathering feedback from optical encoders. It basically handles messages (sent by front-end agents) specifying the *speed* to set for the left and right wheel, forwarding it (after measurement unit conversion) to the motor controller connected through the RS232 line. On the other hand, its periodic activity entails receiving the feedback from optical encoders (i.e. tick count), acquired through another RS232 line, and then computing tick frequency, thus evaluating the real speed of the wheels: the obtained value is used to adjust the value(s) sent to motor controller in order to make each wheel to reach the desired speed[6].

The *RS485 Management* agent is responsible for driving two external boards connected, to the PC, through the same RS485 serial bus: a controller for servo-motors and a board offering a certain number of I/O digital lines. Since each servo-motor and each I/O line is then used by different agents, the RS485 Management acts as a de-/multiplexer for actions and sensed data. Its periodic activity is the sampling of digital inputs, by means of a request/reply transaction through serial messages exchanged with the I/O board; polled data are thus stored in the agent's state in order to make them available for requests coming from other agents. In addition, the RS485 Management is able to receive messages containing commands to be sent to servo-motor, through the servo-controller; in particular, each command specifies the servo-motor to drive and the rotation angle to be set.

The *Start/Stop Control* agent is a reactive one that periodically queries the RS485 Management in order to check if the "start" or "stop" buttons have been pushed. On this basis, it sends appropriate start/stop messages to the Strategy agent (see below) in order activate (resp. block) its behaviour when a match begins (resp. ends). Since the duration of a match is fixed (90 seconds), this agent embeds also a timer that, armed after a start, automatically sends a stop message when the 90 seconds are due.

The *Ball Control* agent is responsible for managing the ball sucking system, the buffer and the ball release system. During its periodic activity, it queries the RS485 Management agent in order to check the input lines signalling that a new ball has been sucked: if this event occurs, on the basis of the colour of the ball[7], it drives the sucking system's arm servo-motor in order to put the ball in the buffer—if the ball is white and the buffer is not full—or to throw the ball away—if the ball is black or the buffer is full. This agent also holds the number of balls in the buffer, information that, queried by the Strategy agent, is used by the latter to control robot behaviour. As for ball release, the Ball Control agent, following a proper command message, is able to interact with the RS485 Management agent and thus drive the servo-motor controlling the release of a ball. Finally, by checking the status of another input digital line, the Ball Control agent is able to understand if a released ball has been successfully put into a hole.

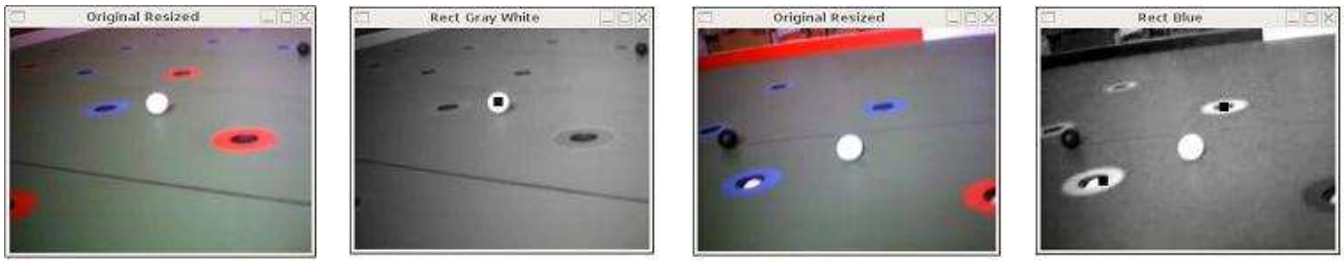
The last agent of the back-end, the *Hole Detector*, reads, through a proper interaction with the RS485 Management agent, the data coming from proximity sensors placed under the robot for hole detection and positioning. It is able to understand the position of the robot, with respect to the hole to catch, and can thus forward this information to the Strategy agent, which, in turn, will drive the wheels to centre the hole and put the ball into it.

C. The Front-End

The front-end layer implements the high-level activities that drive the robot to reach its goal, i.e. placing the most quantity

[6]This is obtained by means of a proportional-integrative-derivative software controller.

[7]The colour is detected through a sensor connected to another digital input line.



(a) (b) (c) (d)

Fig. 4. Recognition by Object Detector agent

of balls into its holes. This layer is composed of three agents: *Object Detector*, *Motion Control* and *Strategy*.

The *Object Detector* has the task of observing the playing area, by means of a USB camera, detecting the objects needed for the game, i.e. balls and holes, and computing their coordinates with respect to the robot position. Since it uses a computation-intensive image manipulation algorithm, this is the sole agent written in C and not in Erlang[8]. This algorithm, whose execution is triggered by a suited message sent by the Strategy agent, exploits artificial vision techniques and performs a series of transformation (i.e. filtering, threshold, binarisation) on RGB planes of each frame acquired in order to isolate and recognise the required objects. Figure 4 reports some screen-shots of the functioning of the Object Detector. In particular, Figures 4a and 4c show two acquired frames, while Figures 4b and 4d illustrate the filtered images with the objects (respectively a white ball and two blue holes) detected by the agent.

The *Motion Control* agent, which is the only PeriodicFSM type, has the task of controlling the robot's path: it receives, from the Strategy agent, messages containing commands for robot positioning, such as *go to X,Y* or *rotate T*, computes the speed of the wheels needed to reach the target, and sends such speeds to the Motion Driver agents. Moreover, in order to ensure that the target is reached, the Motion Control agent periodically requests to Motion Driver the tick count of optical encoders and calculates the absolute position and orientation of the robot [7]. These values are thus compared with the target, making subsequent speed adjustment, if necessary[9]. Another task of the Motion Control agent is obstacle detection. Since the robot has no sensors to detect if an obstacle (e.g. the opponent's robot, a totem, etc.) is in front of it, the Motion Control agent checks if there is no wheel movement within a certain time window (given that wheel's speeds are greater than zero); if this is the case, an obstacle exiting algorithm is started, which entails to move the robot backwards and then rotate it.

The last agent, *Strategy*, is the "brain" of the robot. Being a BasicFSM agent, it is responsible of collecting and putting

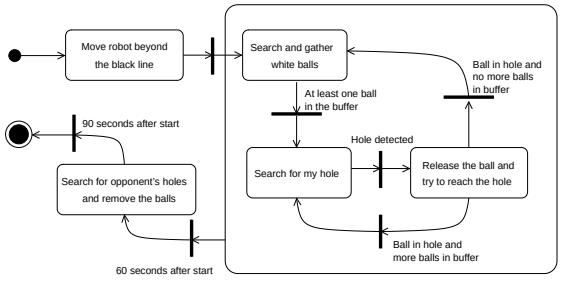


Fig. 5. Strategy Agent Behaviour

together information about environment and robot subsystems to obtain a valuable and effective playing strategy. Even if the field is mostly immutable (except for the position of totems, which are set before each match) and many of the balls involved are still in fixed position, we chose to implements an intelligent and adaptive strategy instead of a simple "fixed–path" one. For this reason the Strategy agent has to adaptively choose the right action to perform at each time, elaborating data coming from other agents. As Figure 5 illustrates, the very first step of the implemented strategy is "move beyond the first black line", since this guarantees the collection of at least one point[10]. This is performed by suitable commands sent to Motion Control agent. When the black line has been passed, the main strategy loop begins. First the robot looks for white balls and suck them into the buffer: if any white ball is seen by the Object Detector, then the Motion Control agent is issued the commands needed to reach the ball; on the other hand, if no ball has been detected, the Strategy agent tries to search elsewhere, by rotating of a random angle in order to look at other zones of the field. When a ball has been sucked and the Ball Control agent reports the presence of at least one white ball into the buffer, the Strategy agent starts to search a right hole to drop it into (i.e. a hole of the colour assigned to the team, either red or blue), looking at messages from Detector and moving toward a hole as soon as it has been found. When the selected hole is no more visible (i.e. outside the camera scope) a ball is released and, by means of messages coming from Hole Detector, a sequence of commands for fine positioning are sent to Motion Control. If the hole is centred

[8]It uses the OpenCV library [2], which provides a set of fast and optimised image manipulation functions. Proper Erlang-to-C library functions allows this agent to interact with Erlang processes.

[9]Also in this case, a proportional-integrative-derivative software controller is employed.

[10]If the robot does not pass the first black line, then it obtains no points at the end of the match.

and the ball goes into it, the Ball Control agent sends a "Ball Successfully Dropped" message, so the Strategy agent decides to search another hole, if more white balls are present into the buffer, or to look for more white balls. If the ball is not dropped into a given amount of time (for example because of errors in fine positioning) the Strategy agent searches another hole and tries to drop the ball into it. Finally, in the last 30 seconds of game, the Strategy agent tries to find opponent's holes to suck white balls out of them.

V. Implementation Issues

As it has been previously said in the paper, with the exception of the Object Detector, the system has been implemented using the Erlang language. However, even if our research group has realized a FIPA-compliant Erlang agent platform (called eXAT [10], [12], [11], [13], [15], [14]), we did not use it in order to avoid overhead introduced by platform's components for inference, behaviour handling, standard FIPA messaging, etc. This is required in order to have a fast and effective support for agents, rather than the possibility of interacting with other external agents (according to Eurobot rules, the robot must be autonomous and not connected to any network). To this aim, each agent of the robot has been encapsulated in an Erlang process and a suitable library has been developed to support the BasicFSM and PeriodicFSM deadline-aware abstractions. Message passing has been realized by means of the native Erlang constructs to perform interprocess communication (which are designed to be very fast): this resulted in an optimised code able to meet to real-time requirements of the target application.

VI. Conclusions

This paper described the architecture of an autonomous mobile robot, developed by the DIIT Team of the University of Catania to participate to the Eurobot competition. A multiagent system has been employed for this purpose, composed of several agents in charge of both interacting with physical sensors and actuators, and supporting the game strategy for the robot. A layered architecture has been designed to clearly separate the aspects above—physical world interface and intelligence—and to favour design, modularity and reuse. Due to real time constraints, the system has been implemented using the Erlang language by means of a proper library to support the abstraction needed for using agents in a robotic environment. This allowed us to develop a fast code able to effectively support robot's activities.

VII. Acknowledgements

The authors wish to thank so much all the other components of the Eurobot DIIT Team, who gave a terrific and fundamental contribution in the realization of the robot described in this paper and made this experience not only very useful but also very funny.

These people are **Roberto Di Salvo, Andrea Nicotra, Luca Nicotra, Massimiliano Nicotra, Stefano Palmeri, Francesco Pellegrino, Matteo Pietro Russo, Carmelo Sciuto, Danilo Treffiletti** and **Carmelo Zampaglione**.

Moreover, the authors wish to thank also the official sponsors of the Eurobot DIIT Team, which are **Siatel Srl** (from Catania, Italy) and **Erlang Training & Consulting Ltd**[11] (from London, UK), that, with their support, contributed to make our dream real.

References

[1] "http://www.erlang.org. Erlang Language Home Page," 2004.
[2] "http://opencvlibrary.sourceforge.net/," 2006.
[3] J. Armstrong, B. Dacker, R. Virding, and M. Williams, "Implementing a Functional Language for Highly Parallel Real Time Applications," 1992.
[4] J. L. Armstrong, "The development of Erlang," in *Proceedings of the ACM SIGPLAN International Conference on Functional Programming*, A. Press, Ed., 1997, pp. 196–203.
[5] J. L. Armstrong, M. C. Williams, C. Wikstrom, and S. C. Virding, *Concurrent Programming in Erlang, 2nd Edition*. Prentice-Hall, 1995.
[6] Bollella, Gosling, Brosgol, Dibble, Furr, Hardin, and Turnbull, *The Real-Time Specification for Java*. Addison-Wesley, 2000.
[7] J. Borenstein, H. R. Everett, and L. Feng, *Where am I? — Systems and Methods for Mobile Robot Positioning*. WWW, University of Michigan, USA, http://www-personal.engin.umich.edu/~johannb/position.htm, 1996.
[8] A. Corsaro and C. Santoro, "Design Patterns for RTSJ Application Development," in *Proceedings of* 2^{nd} *JTRES 2004 Workshop, OTM'04 Federated Conferences*. LNCS 3292, Springer, Oct. 25-29 2004, pp. 394–405.
[9] A. Di Stefano, F. Gangemi, and C. Santoro, "ERESYE: Artificial Intelligence in Erlang Programs," in *Erlang Workshop at 2005 Intl. ACM Conference on Functional Programming (ICFP 2005)*, Tallinn, Estonia, 25 Sept. 2005.
[10] A. Di Stefano and C. Santoro, "eXAT: an Experimental Tool for Programming Multi-Agent Systems in Erlang," in *AI*IA/TABOO Joint Workshop on Objects and Agents (WOA 2003)*, Villasimius, CA, Italy, 10–11 Sept. 2003.
[11] ——, "eXAT: A Platform to Develop Erlang Agents," in *Agent Exhibition Workshop at Net.ObjectDays 2004*, Erfurt, Germany, 27–30 Sept. 2004.
[12] ——, "Designing Collaborative Agents with eXAT," in *ACEC 2004 Workshop at WETICE 2004*, Modena, Italy, 14–16 June 2004.
[13] ——, "On the use of Erlang as a Promising Language to Develop Agent Systems," in *AI*IA/TABOO Joint Workshop on Objects and Agents (WOA 2004)*, Torino, Italy, 29–30 Nov. 2004.
[14] ——, "Supporting Agent Development in Erlang through the eXAT Platform," in *Software Agent-Based Applications, Platforms and Development Kits*. Whitestein Technologies, 2005.
[15] ——, "Using the Erlang Language for Multi-Agent Systems Implementation," in *2005 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT'05)*, Compiégne, France, 19–22 Sept. 2005.
[16] P. Dibble, *Real-Time Java Platform Programming*. Prentice Hall PTR, 2002.
[17] I. Infantino, M. Cossentino, and A. Chella, "An agent based multilevel architecture for robotics vision systems." in *Proceedings of the International Conference on Artificial Intelligence, IC-AI '02, June 24 - 27, 2002, Las Vegas, Nevada, USA, Volume 1*, 2002, pp. 386–390.
[18] E. Johansson, M. Pettersson, and K. Sagonas, "A High Performance Erlang System," in 2^{nd} *International Conference on Principles and Practice of Declarative Programming (PPDP 2000)*, Sept. 20–22 2000.
[19] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," *JACM*, vol. 20, no. 1, pp. 46–61, Jan. 1973.
[20] Liu, J. W. S., *Real-Time Systems*. Prentice Hall, 2000.
[21] C. Varela, C. Abalde, L. Castro, and J. Gulias, "On Modelling Agent Systems with Erlang," in 3^{rd} *ACM SIGPLAN Erlang Workshop*, Snowbird, Utah, USA, 22 Sept. 2004.
[22] A. Wellings, *Concurrent and Real-Time Programming in Java*. Wiley, 2004.

[11] `http://www.erlang-consulting.com`