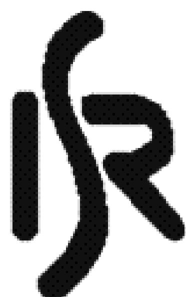
ISocRob-2000: Technical Report

• Rodrigo Ventura Filipe Toscano
Carlos Marques Luís Custódio Pedro Lima •

RT-701-00, RT-402-00



Instituto de Sistemas e Robótica

Pólo de Lisboa

ISocRob-2000: Technical Report

Rodrigo Ventura   Filipe Toscano
Carlos Marques   Luís Custódio   Pedro Lima

October 2000

RT-701-00, RT-402-00

ISR — Torre Norte
Av. Rovisco Pais, 1
1049-001 Lisboa
PORTUGAL

This work was supported by M&T/FESTO (Mota e Teixeira), FCT, IST, ISR, and Agência de Inovação.

Abstract

This technical report describes the status of the robotic soccer team ISocRob (RoboCup initiative, mid-size league) as after the EuRoboCup-2000 event that took place from May 28th to June 2nd at Amsterdam, Netherlands. This report focus on the technical aspects of the robots, in terms of software and hardware.

Keywords: Cooperative Robotics, Multi-Agent Systems, Distributed Artificial Intelligence, Robotic Soccer

Contents

1 Introduction **2**

2 Hardware Issues **2**

2.1 Robot dimensions 3
2.2 Video cameras 5
2.3 Kicker device 8
2.4 Network infrastructure 8

3 Software Issues **9**

3.1 Hardware Abstraction Layer (HAL) 11
3.2 μAgents 13
3.2.1 Vision μAgent 13
3.2.2 Machine μAgent 17
3.2.3 Guidance μAgent 21
3.2.4 Kicker μAgent 22
3.2.5 Proxy μAgent 22
3.2.6 Relay μAgent 22
3.2.7 Monitor and Monitor-X11 μAgents 22

4 Conclusions **23**

References **24**

1 Introduction

This technical report describes the status of the robotic soccer team ISocRob (RoboCup initiative, mid-size league) as after the EuRoboCup-2000 event that took place from May 28th to June 2nd at Amsterdam, Netherlands. This report focus on the technical aspects of the robots, in terms of software and hardware.

The participation of the ISocRob team in RoboCup scientific events is part of the SocRob project running at ISR/IST. This project goal is the design and development of multi-agent systems and the study of methodologies for teamwork among physical robots. The ISocRob team is an appealing application of the methodologies discussed in the context of the SocRob project.

This report is organized as follows: Section 2 describes the underlying conceptual architecture, Section 3 describes the robots hardware , Section 4 focus on the software architecture, and finally Section 5 closes this report with some conclusions about the SocRob project.

The reference architecture of the SocRob project is based on the multi-agent architecture developed by Alex Drougol [3, 2]. The society of agents is divided in three layers: **Organizational**, issues related with the *whole* society; **Relational**, issues related with interactions among two or more members, and which *do not directly affect* the society as a whole; and **Individual**, issues related with individual members. Instantiating this architecture to the robotic soccer team, the organizational layer comprises issues such as determing the team formation or acting upon the start and end of the game, the relational layer comprises for instance the coordination preventing two robots from approaching the ball at the same time, and the individual layer, the issues such as the visual servoing control behind the behavior of dribbling the ball throughout the field.

2 Hardware Issues

The robots are based on `Nomadic SuperScout` robots equipped with the following items:

- Two-wheel differential drive;
- Sixteen sonar sensors radially distributed around the robot, equally spaced;
- Pentium 233MHz based motherboard (PCM-5862), 64MB of RAM, 8GB of hard drive (laptop model), one PCI and one PC104 bus connectors;

- m68k based daughterboard with three-axis motor controller, sonar and bumper interface, and battery level meters;
- Two 12V batteries, 18Ah capacity.

The following components were added to the robot platform:

- Ultrak KC7500CP color 1/3" CCD PAL camera with a Ultrak KL0412DS lens (4mm, F1.2);
- Omni-directional vision assembly (one MicroVideo MVC26C color CCD camera under a 11cm diameter mirror);
- Pneumatic kicking device, based on Festo components, plus two 0.85l bottles for pressurized air storage (70mm diameter by 220mm length);
- Lucent WaveLAN/IEEE Turbo 11Mbps (Silver) wireless ethernet modem connected through a PC104/PCMCIA bridge;
- Bt848 based (Zoltrix TVmax) frame grabber board, with a S-VHS and a Composite video inputs.

The total weight of each robot is about 30Kg for the goal-keeper and 35Kg for the other ones, which complies with the RoboCup limit of 80Kg, and its height is about 64cm which is below the rule limit of 80cm. The footprint of robot complies with the RoboCup rules, as shown in the next section. According to these rules, there are two geometric restrictions imposed at the convex hull of the robot projection onto the floor plane, with all actuators extended to their maximum reach: the area must not exceed 2025cm^2, and any single cut must be no longer than 63cm.

2.1 Robot dimensions

The goal-keeper robot is physically different from the other robots, hence its dimensions must be separately calculated.

To derive the maximum convex hull area (A_{max}) of the field robots (*i.e.,* not the goal-keeper) the following geometrical considerations were used:

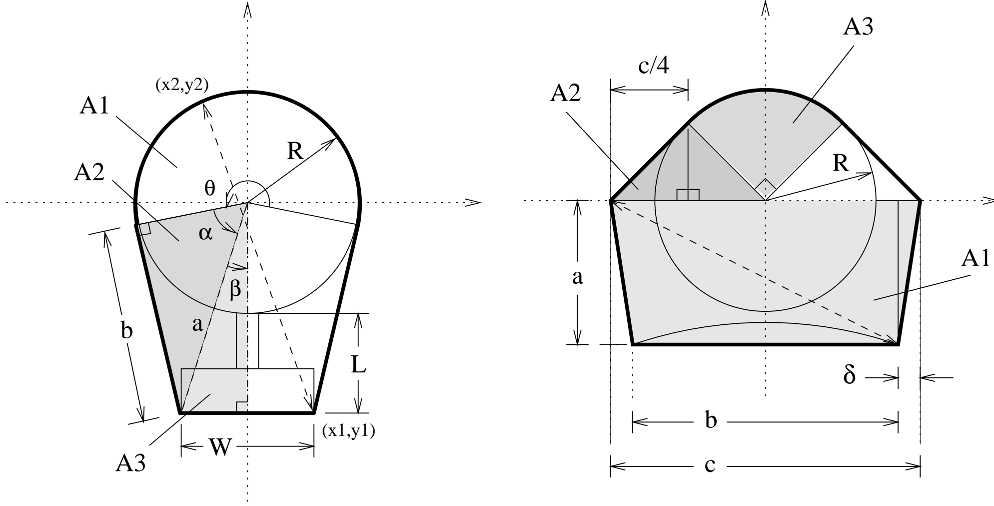


Figure 1: Diagram showing dimensions used for the convex hull footprint calculation: the central circle denotes the SuperScout platform body while the structure attached below denotes the kicking device fully extended (field robot). Left: field robot; right: goal-keeper robot.

$$\begin{cases} a = \sqrt{(L+R)^2 + (W/2)^2} \\ \beta = \arctan \frac{W}{2(L+R)} \\ \alpha = \arccos \frac{R}{a} \\ \theta = 2\pi - 2\alpha - 2\beta \\ b = a \sin \alpha \\ A_1 = \frac{\theta R^2}{2} \\ A_2 = \frac{bR}{2} \\ A_3 = \frac{W(L+R)}{4} \\ A_{max} = A_1 + 2A_2 + 2A_3 \end{cases} \tag{1}$$

For the maximum cut derivation C_{max} (dashed line in figure 1, left part), the following formulae can be used:

$$
\begin{cases}
x_1 = -W/2 \\
y_1 = -R - L \\
x_2 = x \\
y_2 = \sqrt{R^2 + x^2} \\
d(x) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \\
C_{max} = \max_x d(x)
\end{cases} \tag{2}
$$

The robot dimensions (figure 1, at left) are $R \simeq 207$mm, $W \simeq 240$mm, and $L \simeq 190$mm. Therefore,

$$
\begin{cases}
A_{max} \simeq 1992\text{cm}^2 < 2025\text{cm}^2 \\
C_{max} \simeq 62.2\text{cm} < 63\text{cm}
\end{cases} \tag{3}
$$

Concerning the goal-keeper, the equations used to derive its area footprint A^{gk}_{max} are (see figure 1, at right):

$$
\begin{cases}
A_1 = a\frac{b+c}{2} \\
A_2 = \left(\frac{c}{4}\right)^2 \\
A_3 = \frac{\pi R^2}{4} \\
A^{gk}_{max} = A_1 + 2A_2 + A_3
\end{cases} \tag{4}
$$

and with respect to the maximum cut derivation C^{gk}_{max} (dashed line in figure 1, right part):

$$
C^{gk}_{max} = \max\left\{\sqrt{a^2 + (c - \delta)^2}, c\right\} \tag{5}
$$

The goal-keeper robot dimensions (figure 1) are $R \simeq 207$mm, $a \simeq 220$mm, $b \simeq 510$mm, $c \simeq 604$mm, and $\delta \simeq 47$mm. Therefore,

$$
\begin{cases}
A^{gk}_{max} \simeq 2018\text{cm}^2 < 2025\text{cm}^2 \\
C^{gk}_{max} \simeq 60.4\text{cm} < 63\text{cm}
\end{cases} \tag{6}
$$

2.2 Video cameras

The robot contains two cameras: one placed slightly behind the front sonar hole (which was previously displaced), and one in the omni-directional vision

assembly. These cameras are called in this document *front* and *up* cameras, respectively.

In order to obtain accurate distance measurements of an object position using the front camera, its inverse projective geometry was derived. Figure 2 shows the notation used in the derivation. The goal is to obtain x and y coordinates (at the floor level) of an object, given the u and v coordinates on the image plane and the camera intrinsic parameters.

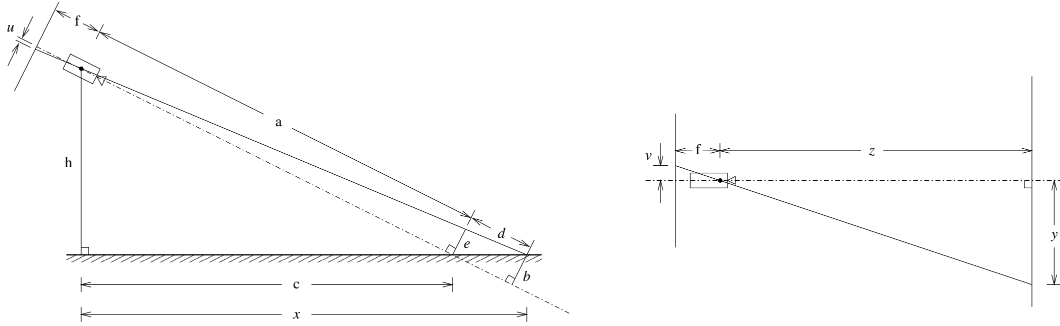


Figure 2: Notation used to derive the front camera inverse projective geometry. Left: sideways view, parallel to the floor. Right: up view, orthogonal to the projection plane and parallel to the wheels axis.

The above geometry suggests the following set of triangle similarity relations:

$$\left\{\begin{array}{l} \frac{h}{a} = \frac{b}{x-c} \\ \frac{a+d}{b} = \frac{a}{e} \\ \frac{e}{a} = \frac{u}{f} \\ \frac{x-c}{d} = \frac{a}{c} \\ \frac{v}{f} = \frac{y}{z} \\ z^2 = h^2 + x^2 \end{array}\right. \tag{7}$$

Taking into account the fact that $a^2 = h^2 + c^2$, it is straightforward to obtain from (7) the desired relations:

$$\left\{\begin{array}{l} x = h\frac{1+\left(\frac{h}{c}\right)\frac{u}{f}}{\left(\frac{h}{c}\right)-\frac{u}{f}} \\ y = \frac{v}{f}\sqrt{h^2 + \left[h\frac{1+\left(\frac{h}{c}\right)\frac{u}{f}}{\left(\frac{h}{c}\right)-\frac{u}{f}}\right]^2} \end{array}\right. \tag{8}$$

The intrinsic parameters of the camera are condensed in three constants: h, f, and c. The camera pitch was adjusted for a convenient[1] $c \simeq 625mm$, while the remaining parameters were obtained experimentally by calibration (solving (8) for a set of calibration points[2]). Note that the image point $(u, v) = (0, 0)$ corresponds to the center of the image. The relevant measurements of objects — the object distance d and angular deviation θ — are easily obtained from (8):

$$\begin{cases} d = \sqrt{x^2 + y^2} \\ \theta = \arctan_2 \frac{y}{x} \end{cases} \tag{9}$$

These equation are only valid for points on the floor level. However, the center-of-mass of the ball in the camera image does not satisfy this constraint. A correction factor has to be introduced for this case (see figure 3). Moreover, the derivation of this correction factor can also be used to evaluate the expected ball radius in the camera image. This value is used to decide whether to consider a given mass of pixels as a ball or just noise.

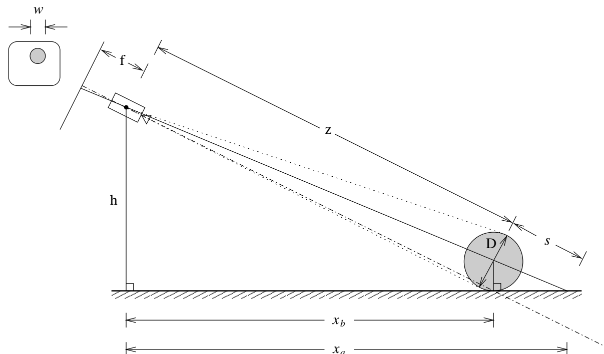


Figure 3: Notation used to derive the correction factor for the ball position measurement.

Figure 3 suggests the following geometrical relations:

$$\begin{cases} \frac{D}{z} = \frac{w}{f} \\ \frac{h}{x_a} = \frac{D/2}{x_a - x_b} \end{cases} \tag{10}$$

[1]The rationale of this adjustment was to have, best visibility of the area close to the robot, whole field visibility, including goals, and visibility below the top of the field walls.

[2]The calibration points were: three points horizontally centered, at 1/4, 1/2 (center), and 3/4 vertical positions.

Solving (10) for the corrected ball coordinate x_b and the expected ball radius in the image w, taking $z \approx z + s = \sqrt{h^2 + x_a^2}$ as a rough estimate (assuming $s << z$), the following expressions can be obtained:

$$\begin{cases} x_b = \left(1 - \frac{D}{2h}\right) x_a \\ w = \frac{Df}{\sqrt{h^2+x_a^2}} \end{cases} \tag{11}$$

The omni-directional vision device is based on a small video camera, pointing upwards. Above this camera stands a mirror with a cylindrical revolution profile so that the camera image equals an orthographic projection of the points at the floor level (apart from an affine transform). The mirror diameter is about 11cm. Further details can be found at the end of Section 3.2.1 and in [7].

2.3 Kicker device

The kicker device is based on a double-effect pneumatic cylinder with a regulated pressure, fed by two pressurized air bottles. The air flow is controlled by an electro-valve. The air circuit schematic can be found in figure 4. The air bottles (d) are externally filled with pressurized air from (a). The valve (c) keeps the air from escaping while the robot is not connected to the external compressor. The pressure regulator (e) feeds the electro-valve (f) with a constant pressure, lower than the one in (d), so as to save air pressure. The electro-valve (f) switches the pressure between the two air entries of the double-effect cylinder (g). Before each game (and during the half-time interval) the bottles are filled with about 8 bar of pressure and the regulator is set to about 4 bar. These settings result in an autonomy of roughly 80 kicks.

The components used in this device were the following Festo components:

- Pression regulator (e), ref. LRMA–1/4–QS–8;
- Electro-valve (f), ref. CPE–10–M1H–5L–M7;
- Compact cylinder (g), ref. ADVUL–20–50–P–A.

2.4 Network infrastructure

Each robot is equipped with a WaveLAN wireless ethernet card (IEEE 802.11 compliant), rated at 11Mbps maximum transfer throughput. These cards use a direct sequence spread spectrum (QPSK) modulation in the frequency band 2400–2483.5MHz. Since no access point was used, the cards were configured

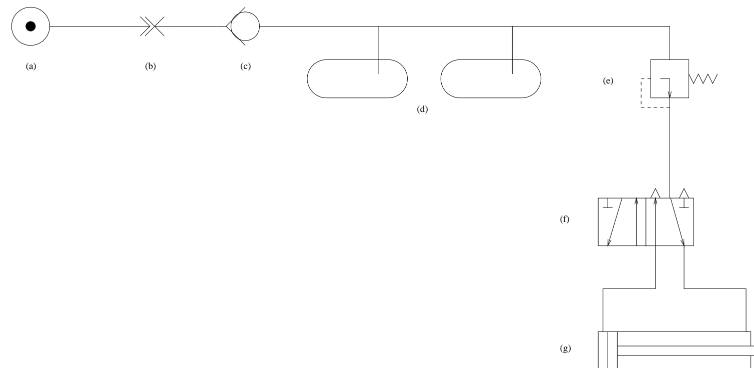


Figure 4: Kicker device pneumatic circuit: (a) external air compressor; (b) connection socket; (c) one-way valve; (d) two 0.85l air bottles; (e) manual (constant) pressure regulator; (f) electro-valve; (g) double-effect cylinder.

to work in *ad-hoc* mode. These cards use a PCMCIA interface, therefore a PCMCIA/PC104 bridge was used (Advantech PCM-3110A).

3 Software Issues

The software architecture shares most of the features used in RoboCup-99 [6]. The architecture was revised while some parts were re-implemented reflecting lessons from previous experience.

As in RoboCup-99, the CVS[3] version management system was used, consisting on a software repository on a server machine, from which each robot obtains a working copy. This way, the software development and the distribution of new code versions among the robots is straightforward. The repository module is called `eurocobup2000`.

Figure 5 shows a diagram of the current software architecture. The modules enclosed by the dotted region labeled HAL[4] constitute the infrastructure on the top of which the main modules work:

Scout — abstracts the Scout robot platform details, providing functions to access the motors controller, odometry, sonars, bumpers, and battery status;

Camera — abstracts the BTTV frame grabber driver details, providing an

[3]**C**oncurrent **V**ersions **S**ystem.
[4]**H**ardware **A**bstraction **L**ayer.

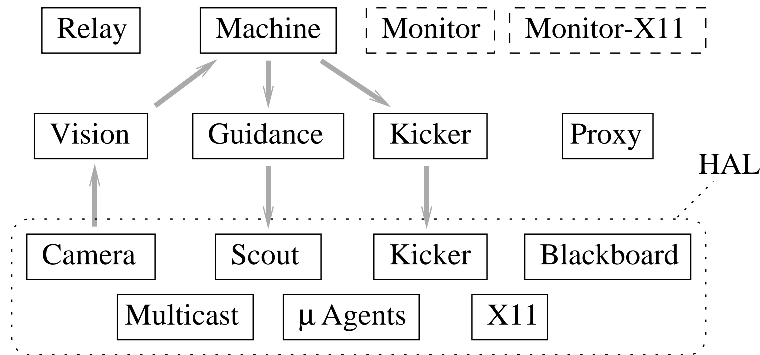


Figure 5: Software architecture: gray arrows denote the main information flow from sensors to effectors.

API[5] to allow full-rate vision processing (through a double-buffering technique);

Blackboard — implements the blackboard, which is the communication medium among the architecture components (intra-robot) as well as between physically distinct robots (inter-robot);

μAgents — implements the μAgents (micro-agents, to distinguish them from the robot as a single agent) facility allowing concurrent processing, abstracting the operating system multi-thread mechanisms;

Kicker — interfaces with the kicker hardware;

Multicast — abstracts the multicast network communication details;

X11 — interfaces with the X Window System (bitmapped display);

All the remaining modules are μAgents constituting the main components of the architecture:

Vision — all vision processing is done by this μAgent, multiplexing the two video inputs (up and front camera), and providing three modes of operation: *front camera, up camera,* and *self-localization;*

Guidance — implements three motion control methods: *position, velocity,* and *potential modes;*

[5]**A**pplication **P**rogramming **I**nterface

Machine — implements a set of state machines which are responsible for the robot behavior. Each state machine is termed a *role*. Three roles were built: attacker, defender, and goal-keeper;

Kicker — interfaces the kicker HAL module, providing the temporization needed for the actuation of this device;

Proxy — waits for networked blackboard variables changes (multicast network packets) updating the local blackboard accordingly;

Relay — socket server providing access to the software architecture, namely blackboard variables access;

Monitor — optional module for text-based remote monitoring;

Monitor-X11 — like the above but X11-based (e.g., camera image monitoring).

The following subsections describe these modules in more detail.

3.1 Hardware Abstraction Layer (HAL)

The source files prefixed with '`hal-`' implement these modules.

The `hal-scout.c` contains functions to access the Scout robot platform. These functions are: *motor control*, in position or velocity mode; *odometry*, providing a posture state vector (x, y, θ) obtained from the integration of the wheel encoder readings; *sonars*, providing a vector containing a proximity value given by each sonar, and allowing the user to restrict the active sonars to an arbitrary subset; *bumpers*, indicating whether there was a collision (ignored because they were totally unreliable); *battery status*, providing a high/medium/low status report.

The `hal-camera.c` implements the interface to the frame grabber allowing channel switching between the up and front camera, and individual frame grabbing. This module handles double buffering internally to allow full-rate image processing.

The blackboard (`hal-blackboard.c`) is a central data structure to the whole architecture, since all modules depend on it for communications. Moreover it also supports the communication among different robots via the network. The blackboard is a set of symbol/value bindings. The symbols are ASCII strings while values can be of one of the following types: integer (`Int`), floating point (`Float`), boolean (`Bool`), and string (`String`). The symbols follow a hierarchical namespace convention, where the levels are separated by a dot. For instance, `local.var.machine.config.def.near-ball` represents

a local (`local`) variable (`var`) related with the machine μAgent (`machine`), containing the configuration parameter `config` of the defender's (`def`) idea of a near ball distance (`near-ball`). The first level can be one of: `local` for variables local to the robot, `*` for variables global (distributed) to all robots, and n for a variable local to a specific robot where n is its ID[6]. The second level corresponds to the type of the variable: `var` for a simple binding between a symbol and a single value, and `talk` for FIFO-like message queue (not used yet, but will be necessary for more complex cooperation mechanisms). Only the first two levels of the hierarchy have built-in meaning. All other levels are somewhat arbitrary. However, coherence is essential for code readability. It was conventioned that the third level corresponds to the μAgent to which a variable is related to, an optional fourth level `config` corresponds to a pre-specified (in the configuration files, described below) configuration parameter, and optional fourth/fifth level corresponding to a possible μAgent sub-module (e.g., guidance method), and a final level describes the variable. Every time a variable is set, a timestamp is attached, to allow detection of outdated values. If the variable is distributed over the network (multicast), the timestamp corresponds to the packet reception time, with respect to the receiver's clock, to avoid confusion due to clock skew among robots. Note that a brief description of all variables in the blackboard can be found in a documentation file[7]. When the software is launched a set of blackboard variables initial values (configuration parameters) from the files `config.dat` and `config-`n`.dat`, by this order, where n is the robot's ID, is read.

The μAgent infrastructure is implemented by `hal-microagent.c` using the OS multi-threading support. This module allows creation and destruction of μAgents and a simple send/receive signal mechanism. Unless the receiver is blocked waiting for a signal, any received signals are ignored.

The `hal-kicker.c` module provides the interface with the kicking device hardware. When ordered to kick, it opens the pneumatic electro-valve during an adjustable time interval (125ms) to provide a both swift and confident kick. The pneumatic electro-valve is connected to the motherboard parallel port, by the means of a high-current driver.

The `hal-multicast.c` abstracts the details of sending and receiving multicast packets over the network. The standardized IP address `224.0.0.1` (RFC1112) is used to broadcast packets to all hosts of a physical network which support multicast.

Finally, the `hal-x11.c` abstracts the Xlib routines for handling the dis-

[6]Each robot has a unique ID (environment variable `ROBOT_ID`, set by the root user `.cshrc` file), being 1 for `scout1`, 2 for `scout2` and so on.

[7]Filename `Docs/Catalog`, with respect to the code base directory.

play of images, namely the ones originating from the vision processing, with some debugging info drawn.

3.2 μAgents

Implementing the robot software as a society of μAgents has demonstrated many benefits, such as modularity, readability, concurrency, information pipelining, to name a few. This section describes each μAgent in further detail. But before proceeding it is important to refer the rationale behind this particular task division among μAgents.

The first implementation of the software in 1998 [1, 8] was based on a big `while (1)` loop containing most of the central code: vision frame grab, image processing, state machine, and motor controller commanding. The first motivation for the introduction of concurrency was to pipeline the vision processing with the remaining stages. In other words, the vision processing should be kept at the maximum rate (the video full-rate if possible), while the other stages are free to use the information provided by vision, at its rate, or do something else (path planning, inter-robot communication, etc.). This principle eventually proved to be very useful and was further extended. The present software architecture contains eight μAgents (while 2 of them are optionally launched and are used for debugging and monitoring purposes only). The following sections describe each μAgent in detail.

3.2.1 Vision μAgent

This μAgent multiplexes vision processing among the three image processing methods, and the corresponding video channel. The multiplexing code resides in `vision.c`. The provided vision processing methods are:

front — this method uses the front camera to search for the ball and both goals in the image. The results are used to update the corresponding blackboard variables, *i.e.,* `local.var.vision.front.*` hierarchy. The projective geometry derived in Section 2.2 is used to obtain the objects relative position to the robot, in world coordinates. The code contained in this method is strongly based on the vision code used in previous competitions [8]. The two core functions are `findBall()` and `findGoal()` that calculates the center of mass of the ball (if any) and goals (if any), given a specified window and color thresholds (further details below);

up — this method uses the up camera to find the ball. On one hand, the up camera field of view is omnidirectional, but on the other hand its range

is limited in distance: the robot body occludes the mirror view of about 1m from the central robot vertical axis, and the range is about 4.5m. Because of the mirror geometry (hardware unwrapping) it is trivial to obtain floor objects positions in world coordinates;

self — This method also uses the up camera and its purpose is to self-localize the robot in world coordinates based on field landmarks. Since the Scout platform provides integrated odometry outputs, this method is used[8] to periodically reset the Scout's odometry integrators.

The color detection is based on thresholds in HSV space. The video frames are captured in `RGB15` format (see the Bt848 datasheet for further details), where each pixel occupies two bytes, "big endian"[9]:

first byte:	$\times$	r_4	r_3	r_2	r_1	r_0	g_4	g_3
second byte:	g_2	g_1	g_0	b_4	b_3	b_2	b_1	b_0

(12)

The red, green, and blue components are encoded in five bits: $R = r_4 \cdots r_0$, $G = g_4 \cdots g_0$, and $B = b_4 \cdots b_0$. Then, each pixel is converted to HSV space (using a lookup table with $2^{15} = 32768$ entries) using [5]:

$$\begin{cases} H = \begin{cases} \frac{G-B}{\max\{R,G,B\}-\min\{R,G,B\}} & \text{if } R = \max\{R,G,B\}, \\ 2 + \frac{B-R}{\max\{R,G,B\}-\min\{R,G,B\}} & \text{if } G = \max\{R,G,B\}, \\ 4 + \frac{R-G}{\max\{R,G,B\}-\min\{R,G,B\}} & \text{otherwise.} \end{cases} \\ S = \frac{\max\{R,G,B\}-\min\{R,G,B\}}{\max\{R,G,B\}} \\ V = \max\{R,G,B\} \end{cases} \tag{13}$$

A color $\mathcal{C}^i$ is defined by a tuple of four thresholds:

$$\mathcal{C}^i = \left\langle H^i_{min}, H^i_{max}, S^i_{min}, V^i_{min} \right\rangle \tag{14}$$

A pixel with color (H, S, V) in HSV space is positively detected as $\mathcal{C}^i$ iff[10] the following constraints are satisfied:

[8]Because of implementation reasons related with code integration issues, it was not possible to get this method working in time for the EuRoboCup-2000 competitions.

[9]The computing jargon term *big endian* means that the most significant bytes precedes the least significant one, in contrast with the opposite *little endian*. In the early times of computing this trivial issue raised a warm debate comparable to the "Gulliver's Travels" Little/Big endian civilizations [endian]. However, modern computers are not yet uniform to this respect, e.g. Intel 80x86, former-DEC Alpha, and ARM CPUs are little endian, while Motorola 680x0, PowerPC, and SPARC are big endian. And to be complete, MIPS CPUs can actually be either, depending on a status bit.

[10]In this document "iff" denotes "if and only if."

$$\begin{cases} H^i_{min} \leq H \leq H^i_{max} \\ S \geq S^i_{min} \\ V \geq V^i_{min} \end{cases} \tag{15}$$

Figure 15 contains a diagram showing these thresholds in HSV space. Note that cylindrical coordinates are used, meaning that the H coordinate is represented as the polar angle around the V axis. The rationale behind this arrangement of thresholds is the following: The human perceptual notion of *color* corresponds to the *hue* (H) component, *i.e.,* all colors with the same hue corresponds to the same perceptual color sensation and vice-versa. Therefore it is natural to select a slice of hue values to detect a given color. On the other hand, pixels with low values of *saturation* (S) and/or *value* (V) are not to be detected since they correspond to colorless pixels and its hue tend to be extremely sensitive to noise. Besides using H^i_{min} and H^i_{max} for color $\mathcal{C}^i$, experience suggested that the minimum values of S and V also depend on which color is to be detected. Therefore S^i_{min} and V^i_{min} are also specific to a $\mathcal{C}^i$. One reason behind this dependence might be the fact that the objects to be detected often have very different predominant values of saturation and value, e.g., the blue goal is seen by the camera much darker than the yellow one. This suggests a dependence between the hue and the other components. This dependence is, for instance, explored by the Fisher linear discriminants [4].

Given a rectangular window in image coordinates, the center of mass is calculated. The objects positions relative to the robot, in world coordinates, are then calculated using equations (8) and (9). For the special case of the ball, the expected ball radius, in image coordinate dimensions, is also calculated using equation (10). This value is used to determine whether a given quantity of detected pixels are to be considered ball or noise. In other words, it is a dynamic threshold depending on the ball distance. For other objects — yellow and blue goals — fixed thresholds for detection were used.

It is important to obtain some additional information about goals other than their center of mass positions. In particular, it is useful to know the distance to the nearest point of the goal and the direction of the most open space (to direct the kick). These calculations are done projecting the detected pixels onto both axis. Specifically, letting $u(x, y) \in \{0, 1\}$ be 1 iff the pixel with image coordinates (x, y) is positively detected as belonging to a goal color, the horizontal $p_h(x)$ and vertical $p_v(y)$ projections are defined by:

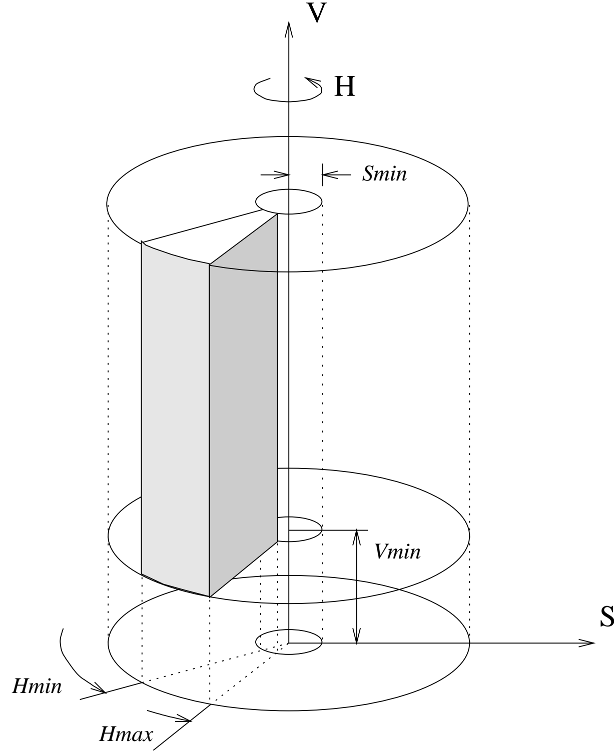


Figure 6: Color thresholds in HSV space, using cylindrical coordinates.

$$p_h(x) = \sum_y u(x, y) \tag{16}$$

$$p_v(y) = \sum_x u(x, y) \tag{17}$$

The nearest point y_n is calculated using $y_n = \underset{y}{\operatorname{argmin}} \{p_v(y) \mid p_v(y) > T_p\}$ with a small threshold T_p for noise robustness. The most open space x_o is a little more complicated to calculate:

$$\begin{cases} x_o = \underset{x}{\operatorname{argmax}} \left\{ \sum_k p_h(k) h(x-k) \right\} \\ h(n) = \begin{cases} 1 & \text{if } -T_o \leq n \leq T_o, \\ 0 & \text{otherwise.} \end{cases} \end{cases} \tag{18}$$

The rationale is to apply a low-pass FIR filter to $p_h(x)$, using the non-causal impulsive response $h(n)$, and then to extract the maximum x_o coordinate.

All these results are posted in the blackboard for usage by other μAgents.

A detailed description of the algorithms used by the self-localization method can be found in [7]. Briefly, this method performs these steps:

1. Detection of green/white transitions, using a strategic sampling of image pixels;

2. Straight lines extraction using the Hough transform;

3. Application of a combinatorial algorithm to select a sub-set of straight lines that satisfy the constraints of the field (right angles, distances between parallel lines, and so on);

4. Sampling of pixels where the goals are expected to be found, in order to disambiguate between field sides[11].

Once the extracted lines are matched against the knowledge of the geometry of the field lines, the robot position is known, apart from which field it is in. This is at last disambiguated by checking the goal colors (step 4).

3.2.2 Machine μAgent

This μAgent is responsible for the robots behavior at several levels. Recalling the conceptual architecture explained in Section 1, this μAgent implements all decision-making in these three levels.

At the organizational level, the team is at a given time in one of these *modes* (or organizational states):

none — idle mode entered when the program starts, where the robot waits for an organizational change of mode (e.g., global post in the blackboard);

play — the team is playing the game against a specified goal (blue or yellow);

pause — entered to suspend temporarily the playing;

setup — the robots move to specified position, usually just before a game start;

test — test mode to accommodate *ad hoc* debugging of specific segments of the code.

[11] Note that the field is symmetric with respect to the middle line, except for goal colors.

Due to lack of time to develop complementary software, the full power of this level was not explored, however the setup/play modes were used when playing and the test mode proved a useful tool to isolate code segments in this complex software architecture.

The relational and individual levels only fully take place in the play mode. In this mode a robot can take one of several available *roles:*

goalkeeper role (`GK`) — uses translational movement (orthogonal to the field's direction) to put itself in an appropriate position;

attacker role (`AT`) — runs a state machine described below, spanning the individual and the relational levels. As to the individual level, it tries to approach the ball and lead it to the opposite goal, possibly kicking it. At the relational level, it implements a basic mechanism to prevent two robots to approach the ball at the same time;

defender role (`DF`) — similar to the attacker one, but for yet unknown reasons (at least at the time of this writing), unable to work ("segmentation fault...");

The goal-keeper uses the up camera exclusively to keep track of the ball, moving along the goal area (in common mode, with the wheels axis parallel to the field direction). The vision processing estimates both the ball position and velocity. Its role is based on three modes of operation: in the FOLLOW mode it follows the ball position, with respect to its movement axis; in the INTERCEPT mode, it moves to the estimated crossing between the robot movement axis and the ball trajectory; and in the HOME mode, it moves to a pre-defined home position (middle of the goal area) after waiting a certain amount of time (for the case of temporary ball occlusion). When the ball is not visible or in the opponent's half-field, the goal-keeper remains in the HOME mode, otherwise it switches between the FOLLOW or INTERCEPT, depending on the degree of danger the estimated ball position/velocity presents to the robot's goal.

The state machine implementing the attacker role is shown in figure 7. Each state is briefly described below:

HOME — during this state the robot is performing a trajectory leading it to the pre-defined home position on the field, by the means of the `ET_go_home()` function;

SHOULD_I_GO — this state serves the purpose of deciding whether it should approach the ball, or not, if there is another robot in better

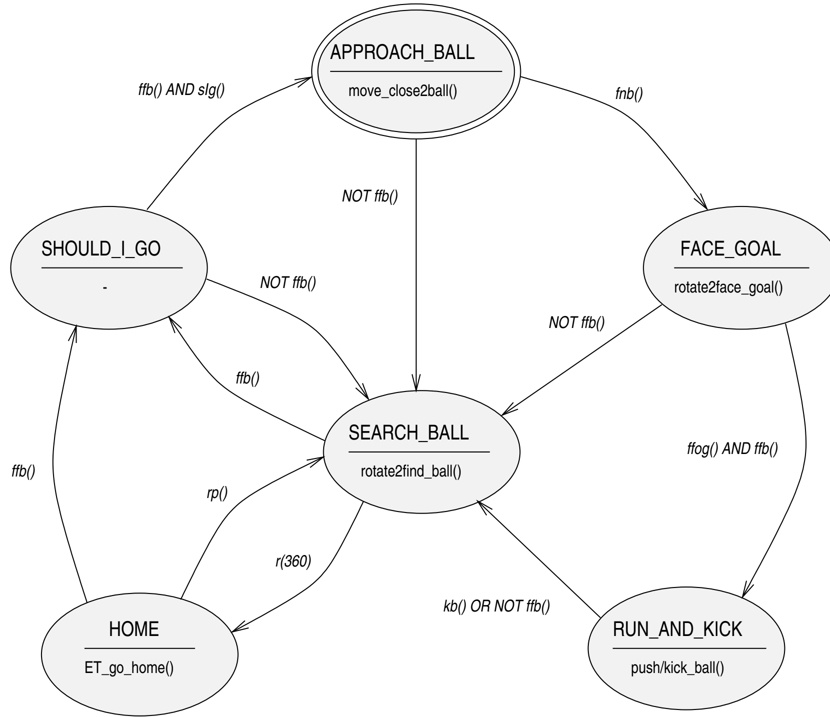


Figure 7: State machine diagram of the attacker role. State names are in caps; the function executed during that state is shown below its name; the transition conditions are predicates (or boolean combinations of them). For the sake of clarity, transitions to the same state were omitted. In other words, given a state, if no transition condition is satisfied, the machine remains in the same state. The initial state (APPROACH_BALL) is denoted by a double-line circle the state name.

condition to do so. This decision is based on inter-robot communication;

APPROACH_BALL — while the robot remains in this state, it will approach the ball, visible by the front camera, using the function `move_close2ball()` to do so;

FACE_GOAL — in this state the robot performs a trajectory in order to face the goal, while holding the ball. This trajectory is a circular arc, with a determined angle (based on current posture and opponent goal relative position), with both common and differential mode components, in order to keep hold of the ball;

RUN_AND_KICK — the robot reaches this state when it holds the ball while facing the opponent goal. If this goal is near enough, it kicks the ball (using the kicking device described in Section 2.3), otherwise it will push the ball towards it. This task is accomplished by the functions `push_ball()` and `kick_ball()`;

SEARCH_BALL — while in this state, the robot rotates around itself for 360 degrees in search for a ball, using the function `rotate2find_ball()`;

The state transitions depend on a set of predicates (or boolean combinations of them):

ffb() — `front_found_ball()`, returns true if the ball is visible by the front camera;

ffog() — `front_found_other_goal()`, returns true if the opponent's goal is visible by the front camera;

fnb() — `front_near_ball()`, returns true if the ball is near the robot, *i.e.,* its distance to the robot is less than a pre-specified threshold (see configuration parameters);

r(360) — `rotated()`, returns true if the robot has already rotated 360 degrees, using the function `rotate2find_ball()`;

rp() — `reached_posture()`, returns true if the robot already reached the desired posture, by the means of the function `ET_go_home()`;

kb() — `kicked_ball()`, returns true if the ball was already kicked (by the kicker);

sIg() — `should_I_go()`, returns true if the robot is the one in better condition to approach the ball. This decision is based on the following: each robot posts to the blackboard (global, network-wide) a value representing the cost of approaching the ball. This value is (periodically) calculated by the function `heuristic()` equaling

$$h = \begin{cases} d_{ball} + k_1 f_{other} & \text{if } f_{ball} = 1, \\ 1000 & \text{otherwise.} \end{cases} \quad (19)$$

where d_{ball} is the estimated ball distance (to the robot), and f_{ball} and f_{other} are 1 or 0 depending in whether the predicates `front_found_ball()` and `front_found_other_goal()` are true or false. The parameter $k_1 <$ 0 weights how much it is favorable to be already facing the opponent's

goal. For instance, if robot 1 and robot 2 are equally close to the ball, if robot 1 is facing the opponent's goal, then it should be the one to approach the ball, while the others stand still. Since each robot's value is distributed over the network, each robot has access to everyone's value. The `should_I_go()` predicate returns true iff the respective robot holds the minimum value at the moment. Therefore (and apart from network latency) it is guaranteed that only one robot has a true `should_I_go()` predicate, and therefore there are never two robots attempting to approach the ball at the same time. Moreover, not only the decision process is instantaneous (no latency for any query-response communication), but also can automatically adjust to changed field situation (e.g., the ball becomes visible to a better positioned robot, that was previously unable to see the ball).

3.2.3 Guidance μAgent

This μAgent is responsible for interfacing the robot motorization. It provides three modes of operation:

velocity mode — given a velocity reference for each wheel in the blackboard, these values are passed to the platform motor controllers, unless an obstacle is found by the sonars. If this is the case, the robot movement is simply halted[12] until further notice;

position mode — this mode is similar to the previous one, except that the references are now given with respect to wheel position (this mode was seldom used);

potential field mode — this mode contains the highest degree of sophistication, since it is based on the well-known principle of potential fields to guide a robot around obstacles. Given an initial robot posture and a final one (with respect to world coordinates), this mode results in a smooth trajectory, where the robot is attracted by the target position and repealed by any obstacle detected by the sonars. Additionally, the ball is considered here as an obstacle (repealing), so that this mode is also able to position itself behind the ball, while not touching it. However, this mode was not frequently used, since its present[13] slowness was not compatible with the competitive level of the games.

[12] This behavior is a source of major headaches, namely during actual games.
[13] Work is underway in order to increase its average speed.

3.2.4 Kicker μAgent

This μAgent monitors the blackboard variable `machine.kicker.kick` so that when it is true, the kicker device (Section 2.3) is activated.

3.2.5 Proxy μAgent

This μAgent listens to a network socket awaiting blackboard messages. When one is received, its contents are posted to the local blackboard. These messages are carried by UDP protocol packets, using multicast addressing. The UDP port used is `2000` and the multicast address is `224.0.0.1` (permanent group of all IP hosts[14], in other words broadcast).

3.2.6 Relay μAgent

The purpose of this μAgent is to provide remote control and monitoring of the software. This μAgent opens a network socket (TCP protocol, port `2001`). When a connection is established, the μAgent responds to a set of commands. The implemented commands are:

`REFR` — Sets the refresh rate for the watched variables;

`ADDW` — Adds a variable to watch;

`DELW` — Deletes a variable watch;

`CLRW` — Removes all variable watches;

`LSTW` — Lists the variables under watch;

`SETV` — Sets a blackboard variable;

`GETV` — Obtains a blackboard variable value;

`QUIT` — Terminates the connection;

3.2.7 Monitor and Monitor-X11 μAgents

These two μAgents provide monitoring of the software for debugging purposes. The Monitor μAgent prints periodically to the console a line of text containing a set of relevant blackboard variables: the current role, the current state, and whether the ball is visible by the up camera. The Monitor-X11 μAgent opens a set X windows, each one showing one vision channel (*i.e.,*

[14] According to the RFC1112 (**R**equest **F**or **C**omments).

camera). The update of each window depends on which camera is active. Furthermore, it draws some debugging information pertaining color detection, center of mass, etc. Each one of these two μAgents can be individually switched on/off.

4 Conclusions

From the scientific point of view, the mid-size league of the RoboCup competition poses two major challenges: first, the playing is a complex, dynamic and highly unpredictable environment, and second, the environment itself was not built by a scientist, in the sense that scientists often tend to tailor the environment in order to show a particular ability of the artifact (s)he is demonstrating.

One drawback of these characteristics is that attention is often deviated to (apparently) minor implementation details, such as mechanical aspects, illumination, video quality, stable power supply, and so on. However, the positive side of the scientific challenge is worthwhile, besides some occasional headaches.

One requirement that the robots have to meet, in order to perform in such environment is that they have to deal simultaneously and coherently with a set of heterogeneous sensors and actuators: vision, sonar, motors, kicking devices. Each one of these devices require a variable amount of complexity on the software side. And since all these software pieces have to work together, their integration becomes a very sensitive issue.

The development of a soccer robot always requires the balance of two objectives: the scientific objective of a clean design, and the pragmatics of winning the games. These goals are not contradictory, but they are seldom satisfied at the same time.

If the software architecture deserves too much attention, there is a risk of devoting too much time writing software that does not qualitatively improve the playing. On the other hand, too much pragmatic programming may overlook the fact that some difficulties are inherent to a poor software design, being virtually impossible to overcome it without a software architecture redesign.

Of course it is wise to devote effort to create a good software architecture, flexible enough to be easily extended to cope with new challenges, and simple enough to provide a smooth integration. However, developments at the level of the architecture building blocks must be carefully evaluated.

References

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

[2] 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.

[3] Alex Drogoul and J. Ferber. Multi-agent simulation as a tool for modeling societies: Application to social differentiation in ant colonies. In *Actes du Workshop MAAMAW'92*, 1992.

[4] Richard O. Duda and Peter E. Hart. *Pattern Classification and Scene Analysis*. Wiley, 1973.

[5] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics*. Addison-Wesley, 1997.

[6] Pedro Lima, Rodrigo Ventura, Pedro Aparício, and Luís Custódio. A functional architecture for a team of fully autonomous cooperative robots. In *Proceedings of RoboCup Workshop of IJCAI-99*. IJCAI, 1999.

[7] Carlos Marques and Pedro Lima. A localization method for a soccer robot using a vision-based omni-directional sensor. In *Proceedings of RoboCup Workshop 2000*, Melbourne, Australia, 2000.

[8] Rodrigo Ventura, Pedro Aparício, and Pedro Lima. RoboCup98 ISocRob team technical reference. Technical Report RT-702-99, ISR, July 1999.