2

CAMBADA soccer team: from robot architecture to multiagent coordination*

António J. R. Neves, José Luís Azevedo, Bernardo Cunha, Nuno Lau, João Silva, Frederico Santos, Gustavo Corrente, Daniel A. Martins, Nuno Figueiredo, Artur Pereira, Luís Almeida, Luís Seabra Lopes, Armando J. Pinho, João Rodrigues and Paulo Pedreiras
Transverse Activity on Intelligent Robotics, IEETA / DETI
University of Aveiro, Portugal

1. Introduction

Robotic soccer is nowadays a popular research domain in the area of multi-robot systems. RoboCup is an international joint project to promote research in artificial intelligence, robotics and related fields. RoboCup chose soccer as the main problem aiming at innovations to be applied for socially relevant problems. It includes several competition leagues, each one with a specific emphasis, some only at software level, others at both hardware and software, with single or multiple agents, cooperative and competitive.
In the context of RoboCup, the Middle Size League (MSL) is one of the most challenging. In this league, each team is composed of up to 5 robots with a maximum size of $50cm \times 50cm$, $80cm$ height and a maximum weight of 40Kg, playing in a field of $18m \times 12m$. The rules of the game are similar to the official FIFA rules, with minor changes required to adapt them for the playing robots
CAMBADA, *Cooperative Autonomous Mobile roBots with Advanced Distributed Architecture*, is the MSL Soccer team from the University of Aveiro. The project started in 2003, coordinated by the Transverse Activity on Intelligent Robotics group of the Institute of Electronic and Telematic Engineering of Aveiro (IEETA). This project involves people working on several areas for building the mechanical structure of the robot, its hardware architecture and controllers (Almeida et al., 2002; Azevedo et al., 2007) and the software development in areas such as image analysis and processing (Caleiro et al., 2007; Cunha et al., 2007; Martins et al., 2008; Neves et al., 2007; 2008), sensor and information fusion (Silva et al., 2008; 2009), reasoning and control (Lau et al., 2008), cooperative sensing approach based on a Real-Time Database (Almeida et al., 2004), communications among robots (Santos et al., 2009; 2007) and the development of an efficient basestation.
The main contribution of this chapter is to present the new advances in the areas described above involving the development of an MSL team of soccer robots, taking the example of the CAMBADA team that won the RoboCup 2008 and attained the third place in the last edition of the MSL tournament at RoboCup 2009. CAMBADA also won the last three editions

*This work was partially supported by project ACORD, Adaptive Coordination of Robotic Teams, FCT/PTDC/EIA/70695/2006.

of the Portuguese Robotics Open 2007-2009, which confirms the efficiency of the proposed architecture.
This chapter is organized as follows. In Section 2 it is presented the layered and modular architecture of the robot's hardware. Section 3 describes the vision system of the robots, starting in the calibration of the several parameters and presenting efficient algorithms for the detection of the colored objects and algorithms for the detection of arbitrary FIFA balls, a current challenge in the MSL. In Section 4 it is presented the process of building the representation of the environment and the algorithms for the integration of the several sources of information received by the robot. Section 5 presents the architecture used in CAMBADA robots to share information between them using a real-time database. Section 6 presents the methodology developed for the communication between robots, using an adaptive TDMA transmission control. In Section 7 it is presented the robots coordination model based on notions like *strategic positioning*, *role* and *formation*. Section 8 presents the Base Station application, responsible for the control of the agents, interpreting and sending high level instructions and monitoring information of the robots. Finally, in Section 9 we draw some conclusions.

2. Hardware architecture

The CAMBADA robots (Fig. 1) were designed and completely built in-house. The baseline for robot construction is a cylindrical envelope, with 485 mm in diameter. The mechanical structure of the players is layered and modular. Each layer can easily be replaced by an equivalent one. The components in the lower layer, namely motors, wheels, batteries and an electromagnetic kicker, are attached to an aluminum plate placed 8 cm above the floor. The second layer contains the control electronics. The third layer contains a laptop computer, at 22.5 cm from the floor, an omni-directional vision system, a frontal camera and an electronic compass, all close to the maximum height of 80 cm. The players are capable of holonomic motion, based on three omni-directional roller wheels.

Fig. 1. Robots used by the CAMBADA MSL robotic soccer team.

The general architecture of the CAMBADA robots has been described in (Almeida et al., 2004; Silva et al., 2005). Basically, the robots follow a biomorphic paradigm, each being centered on a main processing unit (a laptop), the *brain*, which is responsible for the higher-level behavior coordination, i.e. the coordination layer. This main processing unit handles external communication with the other robots and has high bandwidth sensors, typically vision, directly attached to it. Finally, this unit receives low bandwidth sensing information and sends

actuating commands to control the robot attitude by means of a distributed low-level sensing/actuating system (Fig. 2), the *nervous system*.

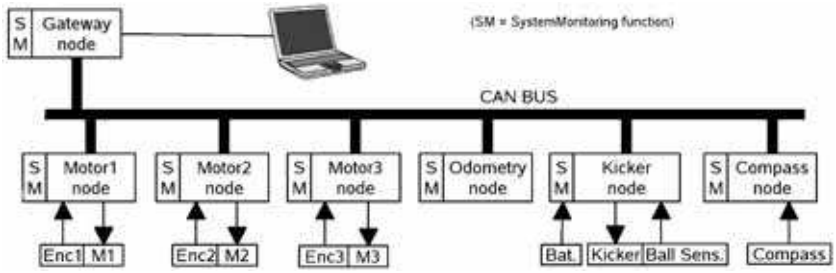


Fig. 2. Hardware architecture with functional mapping.

The low-level sensing/actuating system follows the fine-grain distributed model where most of the elementary functions, e.g. basic reactive behaviors and closed-loop control of complex actuators, are encapsulated in small microcontroller-based nodes interconnected by means of a network. For this purpose, Controller Area Network (CAN), a real-time fieldbus typical in distributed embedded systems, has been chosen. This network is complemented with a higher-level transmission control protocol to enhance its real-time performance, composability and fault-tolerance, namely the FTT-CAN protocol (Flexible Time-Triggered communication over CAN) (Almeida et al., 2002). This protocol keeps all the information of periodic flows within a master node, implemented on another basic module, which works like a maestro triggering tasks and message transmissions.
The low-level sensing/actuation system executes four main functions as described in Fig. 3, namely Motion, Odometry, Kick and System monitoring. The former provides holonomic motion using 3 DC motors. The Odometry function combines the encoder readings from the 3 motors and provides a coherent robot displacement information that is then sent to the coordination layer. The Kick function includes the control of an electromagnetic kicker and of a ball handler to dribble the ball.

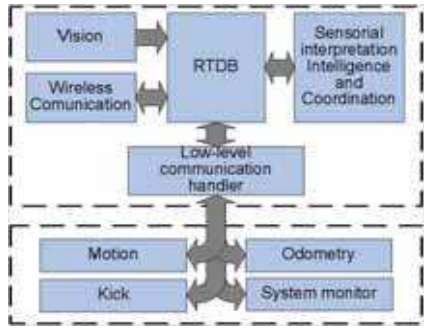


Fig. 3. Layered software architecture of CAMBADA players.

The system monitor function monitors the robot batteries as well as the state of all nodes in the low-level layer. Finally, the low-level control layer connects to the coordination layer through

a gateway, which filters interactions within both layers, passing through the information that is relevant across the layers, only. Such filtering reduces the overhead of handling unnecessary receptions at each layer as well as the network bandwidth usage at the low-level side, thus further reducing mutual interference across the layers.
A detailed description regarding the implementation of this architecture, namely the mapping between the functional architecture onto hardware and the information flows and their synchronization are presented in (Azevedo et al., 2007).

3. Vision system

The vision system of the CAMBADA robots is based on an hybrid system, formed by an omnidirectional and a perspective sub-system, that together can analyze the environment around the robots, both at close and long distances (Neves et al., 2008). The main modules of the vision system are presented in Fig. 4.

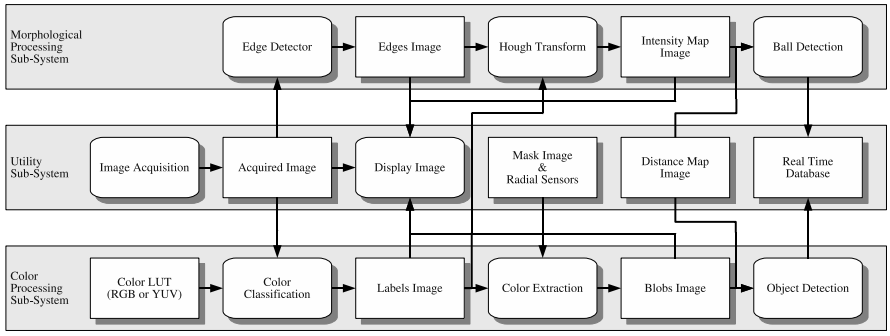


Fig. 4. The software architecture of the vision system developed for the CAMBADA robotic soccer team.

The information regarding close objects, like white lines of the field, other robots and the ball, are acquired through the omnidirectional system, whereas the perspective system is used to locate other robots and the ball at long distances, which are difficult to detect using the omnidirectional vision system.

3.1 Inverse distance map

The use of a catadioptric omni-directional vision system based on a regular video camera pointed at a hyperbolic mirror is a common solution for the main sensorial element found in a significant number of autonomous mobile robot applications. For most practical applications, this setup requires the translation of the planar field of view, at the camera sensor plane, into real world coordinates at the ground plane, using the robot as the center of this system. In order to simplify this non-linear transformation, most practical solutions adopted in real robots choose to create a mechanical geometric setup that ensures a symmetrical solution for the problem by means of single viewpoint (SVP) approach. This, on the other hand, calls for a precise alignment of the four major points comprising the vision setup: the mirror focus, the mirror apex, the lens focus and the center of the image sensor. Furthermore, it also demands

the sensor plane to be both parallel to the ground field and normal to the mirror axis of revolution, and the mirror foci to be coincident with the effective viewpoint and the camera pinhole respectively. Although tempting, this approach requires a precision mechanical setup.
We developed a general solution to calculate the robot centered distances map on non-SVP catadioptric setups, exploring a back-propagation ray-tracing approach and the mathematical properties of the mirror surface. This solution effectively compensates for the misalignment that may result either from a simple mechanical setup or from the use of low cost video cameras. Therefore, precise mechanical alignment and high quality cameras are no longer prerequisites to obtain useful distance maps. The method can also extract most of the required parameters from the acquired image itself, allowing it to be used for self-calibration purposes. In order to allow further trimming of these parameters, two simple image feedback tools have been developed.
The first one creates a reverse mapping of the acquired image into the real world distance map. A fill-in algorithm is used to integrate image data in areas outside pixel mapping on the ground plane. This produces a plane vision from above, allowing visual check of line parallelism and circular asymmetries (Fig. 5). The second generates a visual grid with 0.5m distances between both lines and columns, which is superimposed on the original image. This provides an immediate visual clue for the need of possible further distance correction (Fig. 6).

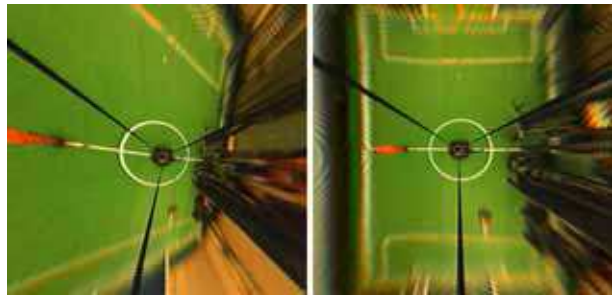


Fig. 5. Acquired image after reverse-mapping into the distance map. On the left, the map was obtained with all misalignment parameters set to zero. On the right, after automatic correction.

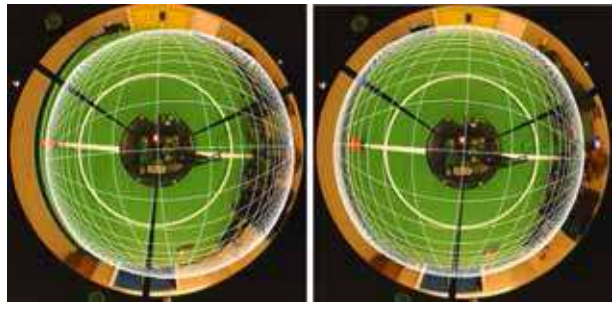


Fig. 6. A 0.5m grid, superimposed on the original image. On the left, with all correction parameters set to zero. On the right, the same grid after geometrical parameter extraction.

With this tool it is also possible to determine some other important parameters, namely the mirror center and the area of the image that will be processed by the object detection algorithms (Fig. 7). A more detailed description of the algorithms can be found in (Cunha et al., 2007).

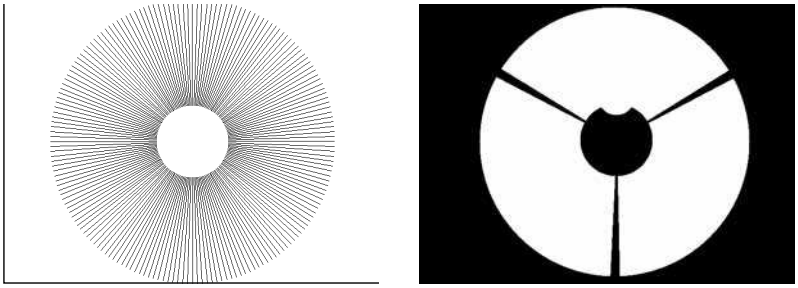


Fig. 7. On the left, the position of the radial search lines used in the omnidirectional vision system. On the right, an example of a robot mask used to select the pixels to be processed by the omnidirectional vision sub-system. White points represent the area that will be processed.

3.2 Autonomous configuration of the digital camera parameters

An algorithm was developed to configure the most important features of the cameras, namely exposure, white-balance, gain and brightness without human intervention (Neves et al., 2009). The self-calibration process for a single robot requires a few seconds, including the time necessary to interact with the application, which is considered fast in comparison to the several minutes needed for manual calibration by an expert user. The experimental results obtained show that the algorithm converges independently of the initial configuration of the camera. Moreover, the images acquired after the proposed calibration algorithm were analyzed using statistical measurements and these confirm that the images have the desired characteristics.

(a) (b) (c)

Fig. 8. An example of the autonomous configuration algorithm obtained starting with all the parameters of the camera set to the maximum value. In (a) the initial image acquired. In (b) the image obtained after applying the autonomous calibration procedure. In (c) a set of graphics representing the evolution of the camera parameters over time.

The proposed approach uses measurements extracted from a digital image to quantify the image quality. A number of typical measurements used in the literature can be computed from the image gray level histogram, namely, the mean (μ), the entropy (E), the absolute

central moment (ACM) and the mean sample value (MSV). These measurements are used to calibrate the exposure and gain. Moreover, the proposed algorithm analyzes a white area in the image to calibrate the white-balance and a black area to calibrate the brightness.

3.3 Object detection

The vision software architecture is based on a distributed paradigm, grouping main tasks in different modules. The software can be split in three main modules, namely the *Utility Sub-System*, the *Color Processing Sub-System* and the *Morphological Processing Sub-System*, as can be seen in Fig. 4. Each one of these sub-systems labels a domain area where their processes fit, as the case of *Acquire Image* and *Display Image* in the *Utility Sub-System*. As can be seen in the *Color Processing Sub-System*, proper color classification and extraction processes were developed, along with an object detection process to extract information, through color analysis, from the acquired image.

Image analysis in the RoboCup domain is simplified, since objects are color coded. This fact is exploited by defining color classes, using a look-up-table (LUT) for fast color classification. The table consists of 16777216 entries (24 bits: 8 bits for red, 8 bits for green and 8 bits for blue), each 8 bits wide, occupying 16 MB in total. The pixel classification is carried out using its color as an index into the table. The color calibration is done in HSV (Hue, Saturation and Value) color space. In the current setup the image is acquired in RGB or YUV format and is then converted to an image of labels using the appropriate LUT.

The image processing software uses radial search lines to analyze the color information. A radial search line is a line that starts in the center of the robot with some angle and ends in the limit of the image. The center of the robot in the omnidirectional subsystem is approximately in the center of the image (an example is presented in Fig. 7), while in the perspective subsystem the center of the robot is in the bottom of the image. The regions of the image that have to be excluded from analysis (such as the robot itself, the sticks that hold the mirror and the areas outside the mirror) are ignored through the use of a previously generated image mask, as described in Section 3.1. The objects of interest (a ball, obstacles and the white lines) are detected through algorithms that, using the color information collected by the radial search lines, calculate the object position and/or their limits in an angular representation (distance and angle). The white lines are detected using an algorithm that, for each search line, finds the transition between green and white pixels. A more detailed description of the algorithms can be found in (Neves et al., 2007; 2008).

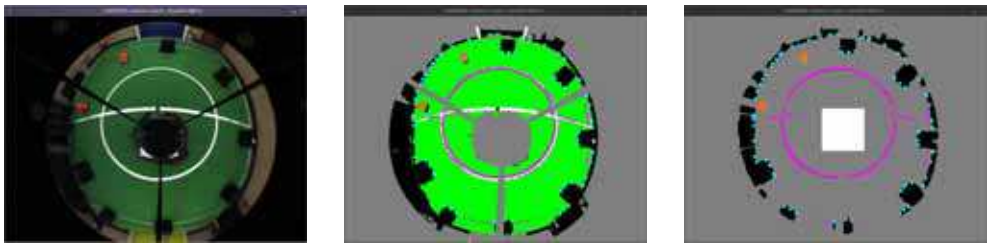


Fig. 9. On the left, the images acquired by the omnidirectional vision system. In the center, the corresponding image of labels. On the right, the color blobs detected in the images. A marks over a ball points to its center of mass. The several marks near the white lines (magenta) are the position of the white lines. Finally, the cyan marks denote the position of the obstacles.

The *Morphological Processing Sub-System* consists of a color independent ball detection algorithm, that will be described in the next section. Martins et al. (2008) presents preliminary results using this approach.
In the final of the image processing pipeline, the position of the detected objects are sent to the real-time database, described later in Section 5, after converting its position in the image into the real position in the environment, using the inverse distance map obtained with the algorithms and tools proposed in (Cunha et al., 2007) and briefly described before.

3.4 Arbitrary ball detection

The arbitrary FIFA ball recognition algorithm is based on the use of edge detection and the circular Hough transform. The search for potential ball candidates is conducted taking advantage of morphological characteristics of the ball (round shape), using a feature extraction technique known as the Hough transform. First used to identify lines in images, the Hough transform has been generalized through the years to identify positions of arbitrary shapes, most commonly circles or ellipses, by a voting procedure (Grimson and Huttenlocher, 1990; Ser and Siu, 1993; Zhang and Liu, 2000).
To feed the Hough transform process, it is necessary a binary image with the edge information of the objects. This image, *Edges Image*, is obtained using an edge detector operator. In the following, we present an explanation of this process and its implementation.
To be possible to use this image processing system in real-time, we implemented efficient data structures to process the image data (Neves et al., 2007; 2008). We used a two-thread approach to perform the most time consuming operations in parallel, namely image segmentation, edge detection and Hough transform, taking advantage of the dual core processor used by the laptop computers of our robots.
The first image processing step in the morphological detection is the edge detection. It must be as efficient and accurate as possible in order not to compromise the efficiency of the whole system. Besides being fast to calculate, the intended resulting image must be absent of noise as much as possible, with well defined contours and be tolerant to the motion blur introduced by the movement of the ball and the robots.
Some popular edge detectors were tested, namely Sobel (Zin et al., 2007; Zou et al., 2006; Zou and Dunsmuir, 1997), Laplace (Blaffert et al., 2000; Zou and Dunsmuir, 1997) and Canny (Canny, 1986). According to our experiments, the Canny edge detector was the most demanding in terms of processing time. Even so, it was fast enough for real-time operation and, because it provided the most effective contours, it was chosen.
The next step in the proposed approach is the use of the Hough transform to find points of interest containing possible circular objects. After finding these points, a validation procedure is used for choosing points containing a ball, according to our characterization. The voting procedure of the Hough transform is carried out in a parameter space. Object candidates are obtained as local maxima of a denoted *Intensity Image* (Fig. 10c)), that is constructed by the *Hough Transform* block (Fig. 4).
Due to the special features of the *Hough* circular transform, a circular object in the *Edges Image* would produce an intense peak in *Intensity Image* corresponding to the center of the object (as can be seen in Fig. 10c)). On the contrary, a non-circular object would produce areas of low intensity in the *Intensity Image*. However, as the ball moves away, its edge circle size decreases. To solve this problem, information about the distance between the robot center and the ball is used to adjust the Hough transform. We use the inverse mapping of our vision system (Cunha et al., 2007) to estimate the radius of the ball as a function of distance.

In some situations, particularly when the ball is not present in the field, false positives might be produced. To solve this problem and improve the ball information reliability, we propose a validation algorithm that discards false positives based on information from the *Intensity Image* and the *Acquired Image*. This validation algorithm is based on two tests against which each ball candidate is put through.
In the first test performed by the validation algorithm, the points with local maximum values in the *Intensity Image* are considered if they are above a distance-dependent threshold. This threshold depends on the distance of the ball candidate to the robot center, decreasing as this distance increases. This first test removes some false ball candidates, leaving a reduced group of points of interest.
Then, a test is made in the *Acquired Image* over each point of interest selected by the previous test. This test is used to eliminate false balls that usually appear in the intersection of the lines of the field and other robots (regions with several contours). To remove these false balls, we analyze a square region of the image centered in the point of interest. We discard this point of interest if the sum of all green pixels is over a certain percentage of the square area. Note that the area of this square depends on the distance of the point of interest to the robot center, decreasing as this distance increases. Choosing a square where the ball fits tightly makes this test very effective, considering that the ball fills over 90% of the square. In both tests, we use threshold values that were obtained experimentally.
Besides the color validation, it is also performed a validation of the morphology of the candidate, more precisely a circularity validation. Here, from the candidate point to the center of the ball, it is performed a search of pixels at a distance r from the center. For each edge found between the expected radius, the number of edges at that distance are determined. By the size of the square which covers the possible ball and the number of edge pixels, it is calculated the edges percentage. If the edges percentage is greater than 70, then the circularity of the candidate is verified.
Figure 10 presents an example of the of the *Morphological Processing Sub-System*. As can be observed, the balls in the *Edges Image* (Fig. 10 b)) have almost circular contours. Figure 10 c) shows the resulting image after applying the circular Hough transform. Notice that the center of the balls present a very high peak when compared to the rest of the image. The ball considered was the closest to the robot due to the fact that it has the high peak in the image.

(a) (b) (c)

Fig. 10. Example of a captured image using the proposed approach. The cross over the ball points out the detected position. In b) the image a), with the Canny edge detector applied. In c), the image b) after applying the circular *Hough* transform.

4. Sensor Fusion

Having the raw information, the Integrator module is responsible for building the representation of the environment. The integration has several sources of information input, being the main input the raw information obtained by the cameras. Besides this information, the integration also uses information given by other sensors, namely an electronic compass (for localization purposes), an infra-red barrier sensor for ball engaged validation, odometry information given by the motors encoders, robot battery status, past cycles worldstate data, shared information obtained from team mate robots and coach information, both concerning game states and team formation, obtained from an external agent acting as a coach.
The first task executed by the integration is the update of the low level internal status, by updating the data structure values concerning battery and infra red ball barrier sensor. This is information that goes directly into the structure, because no treatment or filtering is needed. Afterwards, robot self-localization is made, followed by robot velocity estimation. The ball information is then treated, followed by obstacle treatment. Finally, the game state and any related issue are treated, for example, reset and update of timers, concerning setpieces.

4.1 Localization

Self-localization of the agent is an important issue for a soccer team, as strategic moves and positioning must be defined by positions on the field. In the MSL, the environment is completely known, as every agent knows exactly the layout of the game field. Given the known mapping, the agent has then to locate itself on it.
The CAMBADA team localization algorithm is based on the detected field lines, with fusion information from the odometry sensors and an electronic compass. It is based on the approach described in (Lauer et al., 2006), with some adaptations. It can be seen as an error minimization task, with a derived measure of reliability of the calculated position so that a stochastic sensor fusion process can be applied to increase the estimate accuracy (Lauer et al., 2006).
The idea is to analyze the detected line points, estimating a position, and through an error function describe the fitness of the estimate. This is done by reducing the error of the matching between the detected lines and the known field lines (Fig. 9). The error function must be defined considering the substantial amount of noise that affect the detected line points which would distort the representation estimate (Lauer et al., 2006).
Although the odometry measurement quality is much affected with time, within the reduced cycle times achieved in the application, consecutive readings produce acceptable results and thus, having the visual estimation, it is fused with the odometry values to refine the estimate. This fusion is done based on a Kalman filter for the robot position estimated by odometry and the robot position estimated by visual information. This approach allows the agent to estimate its position even if no visual information is available. However, it is not reliable to use only odometry values to estimate the position for more than a very few cycles, as slidings and frictions on the wheels produce large errors on the estimations in short time.
The visually estimated orientation can be ambiguous, i.e. each point on the soccer field has a symmetric position, relatively to the field center, and the robot detects exactly the same field lines. To disambiguate, an electronic compass is used. The orientation estimated by the robot is compared to the orientation given by the compass and if the error between them is larger than a predefined threshold, actions are taken. If the error is really large, the robot assumes a mirror position. If it is larger than the acceptance threshold, a counter is incremented. This counter forces relocation if it reaches a given threshold.

4.2 Ball integration

Within RoboCup several teams have used Kalman filters for the ball position estimation (Ferrein et al., 2006; Lauer et al., 2005; Marcelino et al., 2003; XU et al., 2006). In (Ferrein et al., 2006) and (Marcelino et al., 2003) several information fusion methods are compared for the integration of the ball position using several observers. In (Ferrein et al., 2006) the authors conclude that the Kalman reset filter shows the best performance.
The information of the ball state (position and velocity) is, perhaps, the most important, as it is the main object of the game and is the base over which most decisions are taken. Thus, its integration has to be as reliable as possible. To accomplish this, a Kalman filter implementation was created to filter the estimated ball position given by the visual information, and a linear regression was applied over filtered positions to estimate its velocity.

4.2.1 Ball position

It is assumed that the ball velocity is constant between cycles. Although that is not true, due to the short time variations between cycles, around 40 milliseconds, and given the noisy environment and measurement errors, it is a rather acceptable model for the ball movement. Thus, no friction is considered to affect the ball, and the model doesn't include any kind of control over the ball. Therefore, given the Kalman filter formulation (described in (Bishop and Welch, 2001)), the assumed state transition model is given by

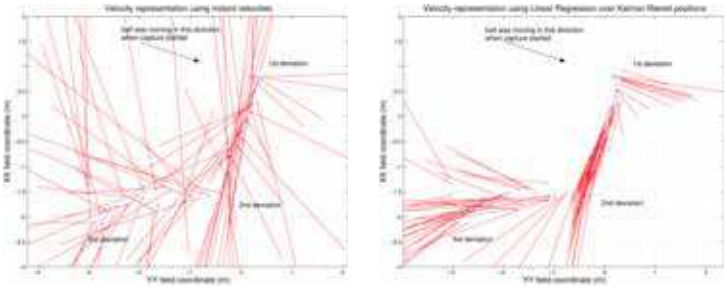
$$X_k = \begin{bmatrix} 1 & \Delta T \\ 0 & 1 \end{bmatrix} X_{k-1}$$

where X_k is the state vector containing the position and velocity of the ball. Technically, there are two vectors of this kind, one for each cartesian dimension (*x*,*y*). This velocity is only internally estimated by the filter, as the robot sensors can only take measurements on the ball position. After defining the state transition model based on the ball movement assumptions described above and the observation model, the description of the measurements and process noises are important issues to attend. The measurements noise can be statistically estimated by taking measurements of a static ball position at known distances.
The standard deviation of those measurements can be used to calculate the variance and thus define the measurements noise parameter.
A relation between the distance of the ball to the robot and the measurements standard deviation can be modeled by a 2nd degree polynomial best fitting the data set in a least-squares sense. Depending on the available data, a polynomial of another degree could be used, but we should always keep in mind the computational weight of increasing complexity.
As for the process noise, this is not trivial to estimate, since there is no way to take independent measurements of the process to estimate its standard deviation. The process noise is represented by a matrix containing the covariances correspondent to the state variable vector. Empirically, one could verify that forcing a near null process noise causes the filter to practically ignore the read measures, leading the filter to emphasize the model prediction. This makes it too smooth and therefore inappropriate. On the other hand, if it is too high, the read measures are taken into too much account and the filter returns the measures themselves.
To face this situation, one have to find a compromise between stability and reaction. Given the nature of the two components of the filter state, position and speed, one may consider that their errors do not correlate.
Because we assume a uniform movement model that we know is not the true nature of the system, we know that the speed calculation of the model is not very accurate. A process

noise covariance matrix was empirically estimated, based on several tests, so that a good smoothness/reactivity relationship was kept.
Using the filter *a-priori* estimation, a system to detect great differences between the expected and read positions was implemented, allowing to detect hard deviations on the ball path.

4.2.2 Ball velocity

The calculation of the ball velocity is a feature becoming more and more important over the time. It allows that better decisions can be implemented based on the ball speed value and direction. Assuming a ball movement model with constant ball velocity between cycles and no friction considered, one could theoretically calculate the ball velocity by simple instantaneous velocity of the ball with the first order derivative of each component $\frac{\Delta D}{\Delta T}$, being ΔD the displacement on consecutive measures and ΔT the time interval between consecutive measures. However, given the noisy environment it is also predictable that this approach would be greatly affected by that noise and thus its results would not be satisfactory (as it is easily visible in Fig. 11.a).
To keep a calculation of the object velocity consistent with its displacement, an implementation of a linear regression algorithm was chosen. This approach based on linear regression (Motulsky and Christopoulos, 2003) is similar to the velocity estimation described in (Lauer et al., 2005). By keeping a buffer of the last m measures of the object position and sampling instant (in this case buffers of 9 samples were used), one can calculate a regression line to fit the positions of the object. Since the object position is composed by two coordinates *(x,y)*, we actually have two linear regression calculations, one for each dimension, although it is made in a transparent way, so the description is presented generally, as if only one dimension was considered.
When applied over the positions estimated by the Kalman filter, the linear regression velocity estimations are much more accurate than the instant velocities calculated by $\frac{\Delta D}{\Delta T}$, as visible in Fig. 11.b).



a) **b)**

Fig. 11. Velocity representation using: In **a)**: consecutive measures displacement; In **b)**: linear regression over Kalman filtered positions.

In order to try to make the regression converge more quickly on deviations of the ball path, a reset feature was implemented, which allows deletion of the older values, keeping only the n most recent ones, allowing a control of the used buffer size. This reset results from the

interaction with the Kalman filter described earlier, which triggers the velocity reset when it detects a hard deviation on the ball path.
Although in this case the Kalman filter internal functioning estimates a velocity, the obtained values were tested to confirm if the linear regression of the ball positions was still needed. Tests showed that the velocity estimated by the Kalman filter has a slower response than the linear regression estimation when deviations occur. Given this, the linear regression was used to estimate the velocity because quickness of convergence was preferred over the slightly smoother approximation of the Kalman filter in the steady state. That is because in the game environment, the ball is very dynamic, it constantly changes its direction and thus a convergence in less than half the cycles is much preferred.

4.2.3 Team ball position sharing

Due to the highly important role that the ball has in a soccer game, when a robot cannot detect it by its own visual sensors (omni or frontal camera), it may still know the position of the ball, through sharing of that knowledge by the other team mates.
The ball data structure include a field with the number of cycles it was not visible by the robot, meaning that the ball position given by the vision sensors can be the "last seen" position. When the ball is not visible for more than a given number of cycles, the robot assumes that it cannot detect the ball on its own. When that is the case, it uses the information of the ball communicated by the other running team mates to know where the ball is. This can be done through a function to get the statistics on a set of positions, mean and standard deviation, to get the mean value of the position of the ball seen by the team mates.
Another approach is to simply use the ball position of the team mate that have more confidence in the detection. Whatever the case, the robot assumes that ball position as its own. When detecting the ball on its own, there is also the need to validate that information. Currently the seen ball is only considered if it is within a given margin inside the field of play as there would be no point in trying to play with a ball outside the field. Fig. 12 illustrates the general ball integration activity diagram.

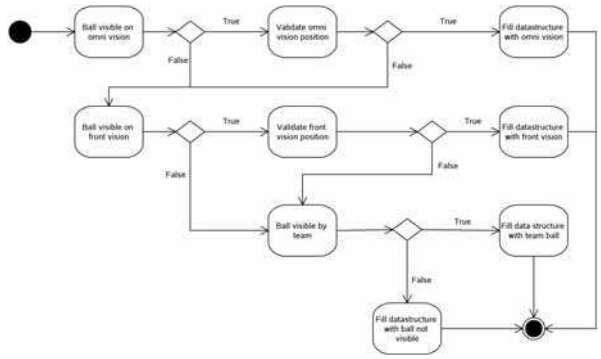


Fig. 12. Ball integration activity diagram.

4.3 Obstacle selection and identification

With the objective of refining the information of the obstacles, and have more meaningful and human readable information, the obstacles are selected and a matching is attempted, in order to try to identify them as team mates or opponents.

Due to the weak precision at long distances, a first selection of the obstacles is made by selecting only the obstacles closer than a given distance as available for identification (currently 5 meters). Also, obstacles that are smaller than 10 centimeters wide or outside the field of play margin are ignored. This is done because the MSL robots are rather big, and in game situations small obstacles are not present inside the field. Also, it would be pointless to pay attention to obstacles that are outside the field of play, since the surrounding environment is completely ignorable for the game development.

To be able to distinguish obstacles, to identify which of them are team mates and which are opponent robots, a fusion between the own visual information of the obstacles and the shared team mates positions is made. By creating a circle around the team mate positions, a matching of the estimated center of visible obstacle is made (Fig. 13), and the obstacle is identified as the corresponding team mate in case of a positive matching (Figs. 14c)). This matching consists on the existence of interception points between the team mate circle and the obstacle circle or if the obstacle center is inside the team mate circle (the obstacle circle can be smaller, and thus no interception points would exist).

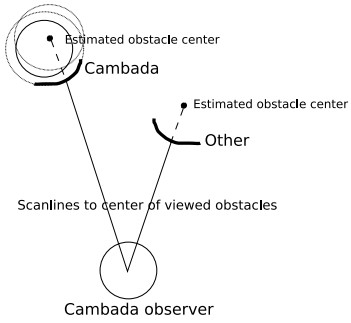
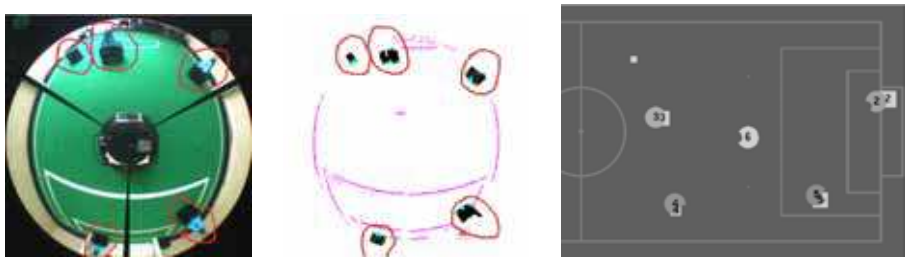


Fig. 13. When a CAMBADA robot is on, the estimated centers of the detected obstacles are compared with the known position of the team mates and tested; the left obstacle is within the CAMBADA acceptance radius, the right one is not.

Since the obstacles detected can be large blobs, the above described identification algorithm cannot be applied directly to the visually detected obstacles. If the detected obstacle fulfills the minimum size requisites already described, it is selected as candidate for being a robot obstacle. Its size is evaluated and classified as robot if it does not exceed the maximum size allowed for MSL robots (MSL Technical Committee 1997-2009, 2008) (Fig. 14a) and 14b)).

If the obstacle exceeds the maximum size of an MSL robot, a division of the obstacle is made, by analyzing its total size and verifying how many robots are in that obstacle. This is a common situation, robots clashing together and thus creating a compact black blob, originating a big obstacle. After completing the division, each obstacle is processed as described before.



a) **b)** **c)**

Fig. 14. Illustration of single obstacles identification. In **a)**: image acquired from the robot camera (obstacles for identification are marked); In **b)**: the same image after processing; In **c)**: image of the control station. Each robot represents itself and robot 6 (the lighter gray) draws all the 5 obstacles evaluated (squares with the same gray scale as itself). All team mates were correctly identified (marked by its corresponding number over the obstacle square) and the opponent is also represented with no number.

5. Real-time database

Similarly to other teams, our team software architecture emphasizes cooperative sensing as a key capability to support the behavioral and decision-making processes in the robotic players. A common technique to achieve cooperative sensing is by means of a blackboard, which is a database where each agent publishes the information that is generated internally and that maybe requested by others. However, typical implementations of this technique seldom account for the temporal validity (coherence) of the contained information with adequate accuracy, since the timing information delivered by general-purpose operating systems such as Linux is rather coarse. This is a problem when robots move fast (e.g. above 1m/s) because their state information degrades faster, too, and temporal validity of state data becomes of the same order of magnitude, or lower, than the operating system timing accuracy.

Another problem of typical implementations is that they are based on the client-server model and thus, when a robot needs a datum, it has to communicate with the server holding the blackboard, introducing an undesirable delay. To avoid this delay, we use two features: firstly, the dissemination of the local state data is carried out using broadcasts, according to the producer-consumer cooperation model, secondly, we replicate the blackboard according to the distributed shared memory model. In this model, each node has local access to all the process state variables that it requires. Those variables that are remote have a local image that is updated automatically by an autonomous communication system (Fig. 15).

We call this replicated blackboard the Real-time Data Base (RTDB), (Almeida et al., 2004) which holds the state data of each agent together with local images of the relevant state data of the other team members. A specialized communication system triggers the required transactions at an adequate rate to guarantee the freshness of the data.

Generally, the information within the RTDB holds the absolute positions and postures of all players, as well as the position of the ball, goal areas and corners in global coordinates. This approach allows a robot to easily use the other robots sensing capabilities to complement its own. For example, if a robot temporarily loses track of the ball, it might use the position of the ball as detected by another robot.

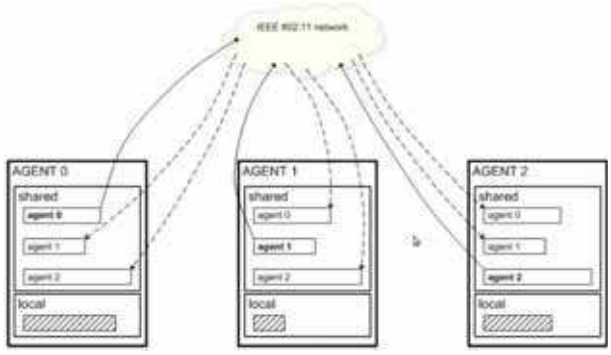


Fig. 15. Each agent broadcasts periodically its subset state data that might be required by other agents.

5.1 RTDB implementation

The RTDB is implemented over a block of shared memory. It contains two main areas: a private area for local information, only, i.e., which is not to be broadcast to other robots; and a shared area with global information. The shared area is further divided into a number of areas, one corresponding to each agent in the team. One of the areas is written by the agent itself and broadcast to the others while the remaining areas are used to store the information received from the other agents.

The allocation of shared memory is carried out by means of a specific function call, `DB_init()`, called once by every Linux process that needs access to the RTDB. The actual allocation is executed only by the first such call. Subsequent calls just return the shared memory block handler and increment a process count. Conversely, the memory space used by the RTDB is freed using the function call `DB_free()` that decreases the process count and, when zero, releases the shared memory block.

The RTDB is accessed concurrently from Linux processes that capture and process images and implement complex behaviors, and from tasks that manage the communication both with the lower-level control layer (through the CAN gateway) and with the other agents (through the wireless interface). The Linux processes access the RTDB with local non-blocking function calls, `DB_put()` and `DB_get()` that allow writing and reading records, respectively. `DB_get()` further requires the specification of the agent from which the item to be read belongs to, in order to identify the respective area in the database.

6. Communications

In the MSL, the agents communicate using an IEEE 802.11 network, sharing a single channel with the opposing team and using managed communication (through the access point), i.e., using a base station, and it is constrained to using a single channel, shared by, at least, both teams in each game. In order to improve the timeliness of the communications, our team uses a further transmission control protocol that minimizes collisions of transmissions within the team. Each robot regularly broadcasts its own data while the remaining ones receive such data and update their local structures. Beyond the robotic agents, there is also a coaching and monitoring station connected to the team that allows following the evolution of the robots status on-line and issuing high level team coordination commands.

As referred above, agents communicate using an IEEE 802.11 network, sharing a single channel with the opposing team and using managed communication (through the access point). This raises several difficulties because the access to the channel cannot be controlled and the available bandwidth is roughly divided by 2.
Therefore, the only alternative left for each team is to adapt to the current channel conditions and reduce access collisions among team members. This is achieved using an adaptive TDMA transmission control, with a predefined round period called team update period (T_{tup}) that sets the responsiveness of the global communication. Within such round, there is one single slot allocated to each team member so that all slots in the round are separated as much as possible (Fig. 16). This allows calculating the target inter-slot period T_{xwin} as $\frac{T_{tup}}{N}$, where N is the number of running agents.

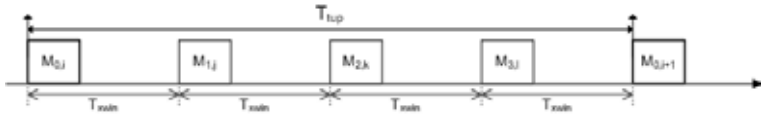


Fig. 16. TDMA round indicating the slots allocated to each robot.

The transmissions generated by each running agent are scheduled within the communication process, according to the production periods specified in the RTDB records. Currently a rate-monotonic scheduler is used. When the respective TDMA slot comes, all currently scheduled transmissions are piggybacked on one single 802.11 frame and sent to the channel. The required synchronization is based on the reception of the frames sent by the other agents during T_{tup}. With the reception instants of those frames and the target inter-slot period T_{xwin} it is possible to generate the next transmission instant. If these delays affect all TDMA frames in a round, then the whole round is delayed from then on, thus its adaptive nature. Figure 17 shows a TDMA round indicating the slots allocated to each agent and the adaptation of the round duration.

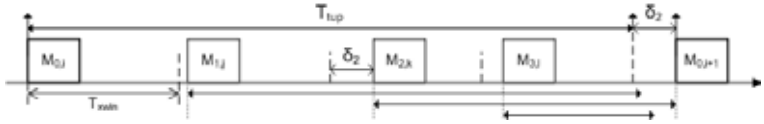


Fig. 17. An adaptive TDMA round.

When a robot transmits at time t_{now} it sets its own transmission instant $t_{next} = t_{now} + T_{tup}$, i.e. one round after. However, it continues monitoring the arrival of the frames from the other robots. When the frame from robot k arrives, the delay δ_k of the effective reception instant with respect to the expected instant is calculated. If this delay is within a validity window $[0, \Delta]$, with Δ being a global configuration parameter, the next transmission instant is delayed according to the longest such delay among the frames received in one round (Fig. 17), i.e.,

$$t_{next} = t_{now} + T_{tup} + max_k(\delta_k)$$

On the other hand, if the reception instant is outside that validity window, or the frame is not received, then (δ_k) is set to 0 and does not contribute to update t_{next}.