Localization issues in the design of a humanoid goalkeeper for the RoboCup SPL using BICA

Víctor Rodríguez, Francisco J. Rodríguez, and Vicente Matellán
Grupo de Robótica, Depto. de Ingeniería Mecánica, Informática y Aeroespacial
E.I.I.I Universidad de León
León, Spain
Email:vrodrm01@estudiantes.unileon.es, fjrodl@unileon.es, vicente.matellan@unileon.es

***Abstract*—This article exposes the localization issues faced during the implementation of a humanoid goalkeeper to take part in the RoboCup SPL standardized soccer robot competition.**

For this task, we have used BICA, a state-driven, component-based architecture created by our counterparts in the URJC, to allow a much easier behavior design process.

The use of BICA let us choose different self-localization methods in configuration time, and the possibility of using none in running time if some type of error appears.

***Keywords*-RoboCup, localization, behavior programming, goalkeeper, Nao**

I. Introduction

The worldwide-known *RoboCup* [7] competition, was established in 1997 to promote research in the robotics field. It is comprised of several events, where *RoboCup Soccer* is the most popular by far. To be precise, RoboCup Soccer has sub-events using different types of robots, ours being *Standard Platform League* (SPL [8]), in which teams must use the same robot model, currently the Nao Humanoid.

The *RoboCup* has gained much recognition because it represents a good opportunity for either testing existing robotic solutions (both software or hardware ones), or developing new ones in a controlled, but not limited, common environment. In our case, we play in SPL soccer. Matches in this competition take place in a well defined environment but it is still complex environment because, while playing, a total of 8 robots (4 per team) will be competing for the ball, trying to score, etc.

As we will see later, the use of a good localization system is mandatory to implement winning strategies in the SPL. Static objects (goals, lines, etc.) on the field may be used as landmarks for the localization system, and almost every team does. But there are many factors like unexpected collisions, visual obstruction by other robots, etc. that could make the self-localization system fail, so robots should be able to fulfill their roles without it.

The BICA architecture [5] was designed to abstract many problems in the development of software control systems for the Nao by offering to the programmer a library of independent and multipurpose components already implemented, letting him or her focus on writing a good behavior instead of reimplementing existing algorithms.

This article describes BICA's multiple localization components, and their use in designing a goalkeeper for the RoboCup SPL. We will show how using BICA facilities the goalkeeper can be designed to use the self-localization methods or not, even to decide during the game if using the localization information or not.

As we will see in next sections all the components are implemented *a priori* and are available at run time. At a given moment the system can choose between them and this will allow to our robot to deploy the better actions in each situation.

The rest of the paper is organized as follows, next section describes the basis of RoboCup regulations, the robot used and the BICA architecture. Section III is devoted to the localization components included in BICA, Section IV to the goalkeeper design and final section to the discussion of the approach.

II. Concepts

A. The environment

All field specifications and match rules can be found in the book that is published every year [4] by RoboCup. The football field where the robots will play has fixed dimensions and layout, the figure 1 shows the dimensions of the field.

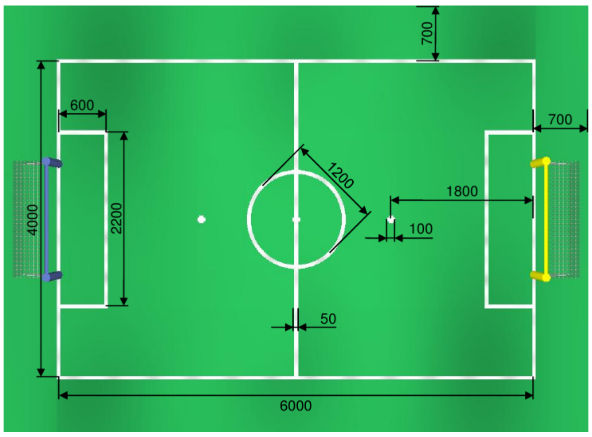

Figure 1. Diagram to scale of the football field used in SPL (dimensions in mm), as seen in the official rulebook.

As it can be seen, the field has a number of lines at specific locations that is used by the localization components to calculate the position of robot on the field. The small choice of colors also allows to easily distinguish between the different elements, like goals. The ball itself is a Mylec orange street hockey ball, 65 mm in diameter, according to the book.

B. The robot

The robot that will be used by all the teams in SPL league is Nao (figure 2), an humanoid robot built by the french Aldebaran Robotics. The full list of features is available in [6], so the advanced technical details will not be discussed here.

The robot has two cameras mounted on its head, but only one can be used at a given moment, and they offer a limited resolution to the detriment of speed. One is located higher and offers a view of the field, and the other is better for controlling the ball as it points lower.

Nao is also equipped with ultrasonic, obstacle-detecting sensors; speakers and microphones for team coordination (via an integrated speech synthesizer); and wireless capabilities, which is the only feasible method for communication while playing.

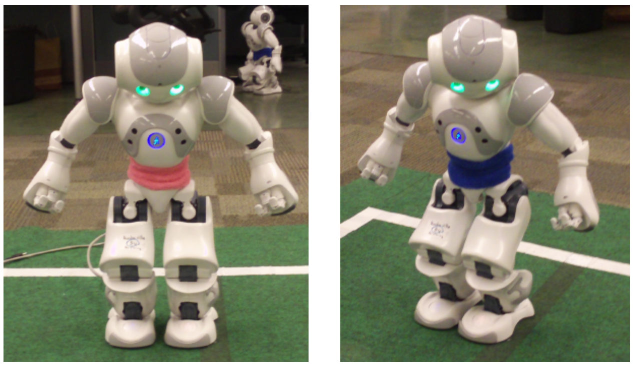

Figure 2. Nao robot depicted with both team markers, as seen in the official rulebook.

Each team will be made up by 4 robots: a goalkeeper and three field players. Colored waistbands must be worn by the robots, to help visual identification. All robots must obey the wireless commands of a state-machine GameController running on a PC controlled by a human referee to signal penalizations to robots, goals, etc.

C. BICA

We did not design our player from scratch. We have used an SDK named BICA that works as a proxy between the framework provided by Aldebaran, and the final application. This SDK, *Behavior-based Iterative Component Architecture* (BICA, [5]), abstracts from many low level details so users can concentrate on writing and debugging medium/high level robot behaviors. BICA has been developed by the Robotics Research group from the University of Rey Juan Carlos of Madrid (URJC), our partners in the SpiTeam.

BICA is a component and state-based software package. It is divided in *Components*, each one providing a different service or feature. For instance, there is a component that takes care of ball detection [2], another for goals detection, several more for global localization, ball kicking, etc. Developers can also create their own components

These components can be loaded or discarded in real time, and BICA can dynamically assign processor power to the active ones. Thanks to this, the amount of power and CPU consumption is kept at minimum, and every component is called as frequently as Nao's hardware is able to manage.

Every cycle, all activated components are stepped in a controlled manner. This is implemented by calling a function in a iterative way. The name of this function is the same in every component: `step()`.

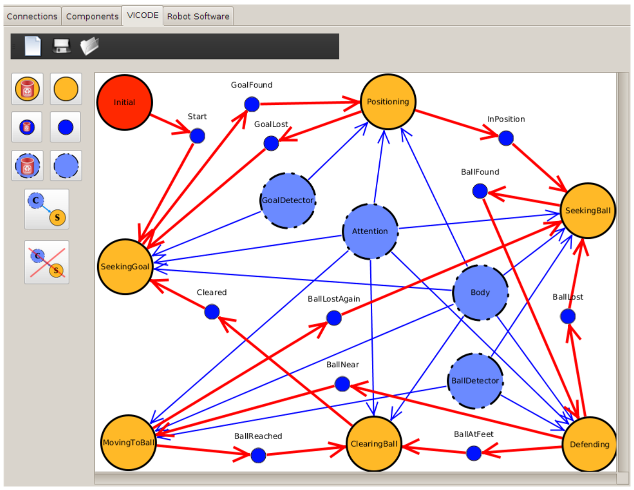

Figure 3. BICA design diagram for a simple goalkeeper behavior. Big yellow circles are *states*, dark blue arrowed spheres are *transitions*. Big, discontinued, purplish blue spheres represent BICA component dependencies.

Components can be organized into hierarchies to implement more sophisticated behaviors. When the robot needs to run a task made up by several components organized in that hierarchy, the `step()` function in the higher component triggers a tree of calls to each `step()` function in the components below. Every component can also request the activation of another non-active one. This way, BICA takes care of stepping dependencies before the code that requires them is stepped on its own.

A BICA component is made up by *states* and *transitions* that connect states. That is, a BICA component is a finite state automata. Each state consists in a piece of code run only when the component has been "stepped", and the state is *active*.

Only one state may be active at any time. The code of the active state will be the one being executed every time the component is stepped. Transitions from the currently active state are evaluated at every step to decide whether to switch to another state or remain in the current one. Transitions are basically boolean decisions, that can be complex functions or just a single boolean variable.

Components can be designed using *VIsual COmponent DEsigner* (VICODE) tool (figure 3). As we can see, left

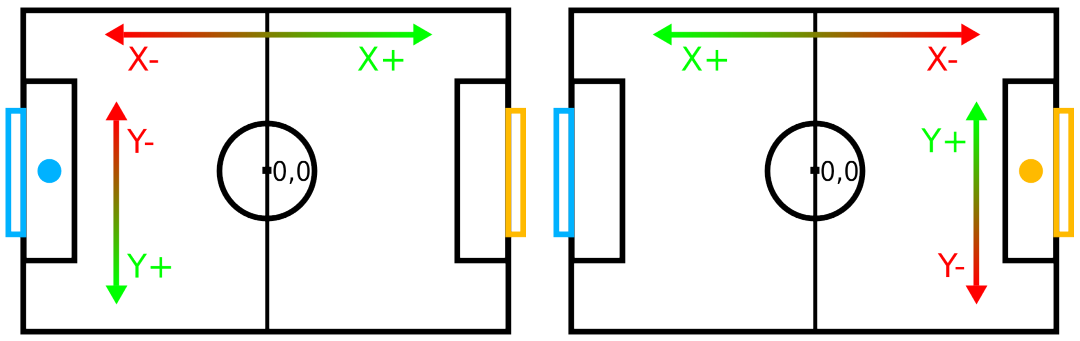

Figure 4. Coordinates given by BICA are team-independent. Position (0,0) marks always the center of the field. Increasing X-axis values get nearer the opponent's goal. Increasing Y-axis values are nearer to the right touchline.

side shows a fixed palette of elements lets us draw and interconnect states, transitions and dependencies . The editor also provides a way to input code directly, and to set automatically timers for time-controlled transitions.

When saving a Component, VICODE automatically generates the corresponding C++ code from the diagram creating the skeleton of C++ classes that can be completed by the designer. Human-written code will be kept the next time the Component is edited. When the code is fully implemented, it will be compiled optimized for the robot CPU and loaded in the robot. Another tool made in Java, named `JManager`, allows us to easily interact with the components running in the robot from our PC.

III. Localization systems

The basic behavior of a goalkeeper is prevent the ball ends inside the goal. To do this, the robot should try to stay under its own goal. The robot could move its body between both posts and take position in front of the ball to do this. This seems simple, but some problems arise:

- What would happen if our robot was to be carried away from the goal? (e.g., grabbed by the referee after a foul, or pushed by another robot)
- Should the robot move away from the goal,to try to clear the ball in case it came near, but not close enough for a direct kick?

The first one is the worst by far. If our robot is not able to navigate through the field, its performance will be very poor. For instance, at the beginning of the match, both teams have 45 seconds to get their position in the allowed areas. If they fail to do so, the referee places them manually on preselected spots, worst than the ones allow by the rules if the robots are able to self-position themselves.

This problem is know as the *kidnapped robot*. Ideally, the localization system should be able to estimate where our robot is in a given moment, and recover quickly after a “kidnapping”.

The second problem is related to the strategy we want to use in the game. If we want to play an offensive match, but our localization system is too weak, we cannot think in kicks far away from our goal position. Whereas if our localization system is well implemented we can deal other movements far from our goal.

BICA has different implementations of localization algorithms, and we use them to get distances and bearings to all interesting components during the game.

To prevent confusions and minimize the code, all methods follow the same conventions concerning the coordinate system. A right-hand system is employed so, regardless of the robot's team, decisions and tactics need not to be tailored for each, as it can be seen in figure 4. In the same way, all localization components follow this coordinate system except the basic sistem, this gets a simple localization from opponet and own goal position.

We have to choose between how the localizations systems are launched. If the choice is going to do it at running time, all the potentially desired localization components have to be loaded and launched int he robot.We will find two problems with this system. If the inner estimation error of each method is going to be used to decide which one is used, all the components have to be active and for this reason all tree of calls for each component (`step()`) are made. This means that it will consumes a lot of processing power to keep all components alive. Other problem, can be find when we stop and activate between localization methods because they have to trigger the system from scratch and this means that a lot of time will be used to initialize components.

Regarding the localization methods included in BICA, there are three different components available:

A. FMK

FMK [14] stands for ”Fuzzy MarKov” and was the localization method developed originally in the TeamChaos, the previous team where our group was integrated before making up SpiTeam.

In FMK, the field is represented by a grid G so that $G(x, y)$ represents the probability of finding the robot at a given position (x, y). The cells contains information of the probability that the robot is in that cell, and information on the most probable orientation range, which means that it is actually a $2\frac{1}{2}$ grid because only one orientation is represented.

This information is represented by a diffuse trapezoid (Fig. 6). This trapezoid is defined by the tuple $(\theta, \delta, \alpha, h, b)$. Intuitively, if h is low, the probability of being in this cell is low. If h is high, it is very probable

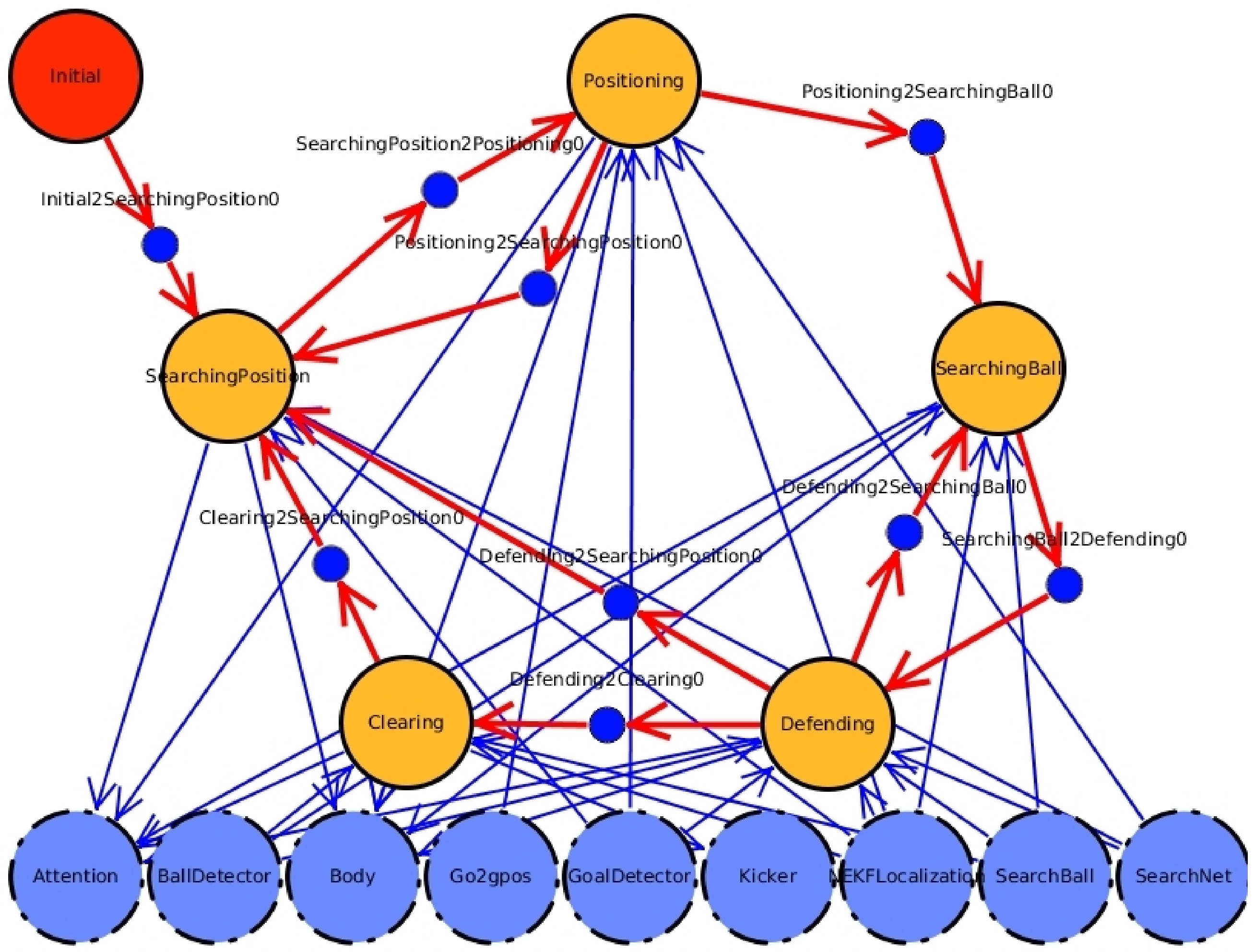

Figure 5. BICA design diagram for a total goalkeeper behavior.

that the robot is in this position. If the trapezoid is wide (is large), great uncertainty exists about the orientation of the robot. If the trapezoid is narrow, or even has triangular form (because is practically null), the orientation uncertainty is so low that we can affirm that the robot orientation is. θ

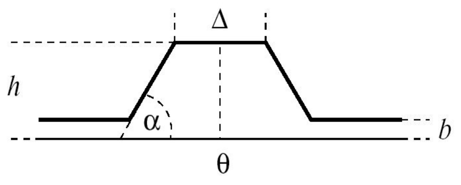

Figure 6. Fuzzy trapezoid

B. n-EKF

The Extended Kalman Filter (EKF) is one the most popular tools for state estimation in robotics [9]. It is a local localization method whose strength relies on its simplicity and its low computational complexity. However, this method is not able to self-locate the robot quickly when starting from a situation of total uncertainty. Neither, is it able to recover from situations of high error in the estimations, nor from the manual change of the position of the robot during the game. This is due to its mono-modality, that is, it only maintains a single estimation of the position of the robot.

Using the n-EKF[12] our team has tried to maintain several hypotheses, each one represented by an independent EKF. The number of EKF filters will not keep constant. It can be dynamically modified up to a maximum limit. Initially, there will not be any active EKF. Every time that the global localization system seems reliable, but no filter close enough to the position estimated by it, a new EKF filter is created. The new EKF filter is initialized to the center of the cell position where the global systems is, and the uncertainty is set to the one associated with that cell.

C. 3D reconstruction

The third method available in BICA is the one based in the 3D reconstruction of the goals [15] through a color based geometrical segmentation method. It uses an HSV color filler, an edge filter, and Hough transformation to detect the post and crossbar lines. Afterwards, the position of the robot with respect to the goal is calculated exploiting

3D geometric properties.

Another methods have also been tested on the our team platform, as for instance [16] based on genetic algorithms, It has not been included in the standard distribution of BICA due to its high computing power requirements.

D. Experimental evaluation of localization algorithms

In this paper we have not tried to describe the insights of the different localization algorithms, but the use of BICA to integrate them, and its use to implement a goalkeeper behavior.

There are several evaluations of the different localization methods for the RoboCup in the literature. For instance, in Hessel bachelor thesis [19] a quick review of all localization methods used in Robocup 2011 can be found, among them we can found SpiTeam localization.

In [12] we can see a depth comparison between the methods available on BICA on the four legged league, the ancestor of the current SPL league.

Finally, regarding Kalman-based methods, in [18] a good review can be found.

Our choice of the localization algorithm is usually made at design time. If we look at the diagram in figure 5 we have all the desired component necessary to deploy the goalkeeper behavior. Two of the dependencies (discontinued, purplish blue spheres) represent the *NEKF* (complex) and the *SearchNet* (basic) localization systems implemented. We load only these ones to save energy consumption and to avoid not covered states where the robot can not find their localization (for instance if the goalkeeper is surrounded by robots).

IV. Role of localization in a RoboCup goalie

Localization ability plays a big role during RoboCup matches. Localization is even more ”necessary” for the goalkeeper than for the other players, otherwise some of its actions would be constrained. Other players do not need to know their exact positions on the field to look for the ball and kicking it towards the opponent goal. But the goalie has to remain into a critical area where a misplacement could cost victory.

The simplest solution would be to add extra transitions from every state, checking if the distance to the goal is greater than a fixed value. Then, the robot should block the ball whenever it came too near (another parameter to be tunned). The algorithm consists in drawing a line between the ball and the center of its goal, follow that line towards the ball, not leaving the area. If the ball gets closer enough it has to kick it out towards the opponent goal.

A more risky situation appears if the ball is close to the goalie, but not enough for kicking, and outside the set *save area* where the goalkeeper can go (the parameter controlling the maximum distance). Ignoring the ball could let an opponent approach it and score a goal. But, if the robot walks out to clear it, the goal will be exposed.

Time constrains are also critical, for instance, if the robot is allow to go to the ball in the previous situation, but if it accidentally moves the ball further away (e.g., with its feet), it can continue walking towards it going far away from the goal. Even if it means to increase the overall complexity, we must make sure that our robot does not “insist” to carry out an action when it is clear that it can not be achieved, or that it has no effect. Some inverse hysteresis has to be considered when defining the states of the component.

Finally, we have not discussed the navigation system. For the goalie we are using a simple control in velocity. We tried to be always faced to the ball, first with the head, then we align the body, so the navigation algorithm is straightforwards. Inside the area the goalkeeper cannot be touched by the opponents, so we do not need to implement an avoidance behavior. There are also no problem to bump into a teammate because they are not allowed to enter its own area (an the referee will remove them if entering). We only should worry about collisions if going out, but the goalie will only go out if it is the closest to the ball, so it has not to worry about penalties.

A. Without localization

If no localization system is implemented, or the readings are not precise, our goalie will have to work without it. This is actually feasible, but the results are less than ideal. If no visual element is present in Nao’s visual field (goals, ball, lines...) in a given moment, the robot is considered lost. This can happen when it is looking in the wrong direction or when another player block our view.

The first approach of our goalkeeper in 2009, was developed without any localization system [17] and was tested in Robocup 2009. The robot implemented a simple system to remain under its own goal bar, taking care of how many steps had run left and right. It also has reactive actions to saving goals when the ball was near the goal.

The other actions of the goalie worked in the same way, except by kicking the ball. The only way to control the movements of the robot would be either to count the number of steps, or the time after leaving the goal, by using a timer and a direction.

B. Using localization

In case of using a localization, the given error in the calculated position is small and the goalie behavior design is much more simple and allow us to do creative play.

Our goalkeeper components are receiving constant data about its current position on the field. BICA has functions that let us specify a precise location where the robot will try to move to. After adjusting manually its bearing (which is also given by the localization system), Nao should place itself quickly under the goalbar. It lets to look at the center of the field, so finding and tracking the ball will be easier.

The approach used in 2011 is to draw two virtual areas. The first one marks the limits that our robot should not pass while defending the goal. The second one, a bigger area should be used when the ball comes too near and the robot gets out of the goal to clear it. If the localization data is reliable, the goalkeeper would get back in position after crossing any area.

V. Conclusion and Further Work

This paper has tried to show the basic problems related to autonomous self-localization of autonomous humanoids in the RoboCup competition. We have shown in particular how the goalkeeper behavior can be implemented with or without localization with BICA framework and how the use of localization is a must if a competitive behavior is desired. We have also summarized the localization methods included in the standard distribution of our software.

Regarding further work, an empirical evaluation of the different methods provided by BICA has not been done in real game conditions. This requires the design of a rigorous methodology for evaluating localization in different situations, in particular, this require offline tools for recording a match a reproducing the same conditions using a different localization component. Anyway, as the information provided by the localization is used to take decisions it is not possible to reproduce exactly the same conditions.

Also we would like to add a different navigation skills, for instance walking backwards facing the ball, or more in general, a positional navigation while tracking the ball.

Acknowledgment

We want to express our gratitude to the work developed by the SpiTeam SPL Team, and specially to Dr. Francisco Martín for his work on localization, Dr. Carlos Agüero for his work on BICA, and Dr. José María Cañas for the general design of the architecture, as well as to the rest of our teammates for their contributions.
This work has been partially supported by Cátedra Telefónica - Universidad de León (grant CTULE11-2).

References

[1] Francisco Martín Rico, Carlos Agüero Durán, José María Cañas , Eduardo Perdices, *Humanoid soccer player design*. In "Robot Soccer". Edited by: Vladan Papic, pp 67-100. 2010. ISBN 978-953-307-036-0.

[2] Francisco Martín Rico, Carlos Agüero Durán, José María Cañas *Follow ball behavior for an humanoid soccer player*. X Workshop de Agentes Físicos, Cáceres, September de 2009.

[3] José María Cañas, Domenec Puig, Eduardo Perdices, Tomás González, *Visual goal detection for the RoboCup Standard Platform League*. Available; http://www.robotica-urjc.es/publicaciones/waf2009-goals.pdf, X Workshop de Agentes Físicos, Cáceres, September de 2009.[Aug, 14, 2011].

[4] RoboCup Technical Committee, *RoboCup Standard Platform League (Nao) Rule Book*. Available: http://www.tzi.de/spl/pub/Website/Downloads/Rules2011.pdf, Germany, March 23, 2011.[Aug, 10, 2011].

[5] Francisco Martín Rico, *Behavior-based Iterative Component Architecture*. Available: http://www.robotica-urjc.es/apuntes/BICA.pdf, Móstoles, Spain, December 15, 2009, [Aug, 20, 2011].

[6] Aldebaran Robotics. *Nao Academics Data Sheet*. Available: http://www.aldebaran-robotics.com/en/node/1166, France, 2011, [Aug, 20, 2011].

[7] RoboCup. Internet: http://www.robocup.org/,[Aug, 23, 2011]

[8] Standard Platform League. Internet: http://www.tzi.de/spl/bin/view/Website/WebHome,[Aug, 20, 2011].

[9] Sebastian Thrun, Wolfram Burgard, Dieter Fox, *Probabilistic Robotics*, MIT Press, ISBN: 0262201623, 2005.

[10] José María Cañas y Vicente Matellán. *From Bio-inspired vs. Psycho-inspired to Etho-inspired robots*. Robotics and Autonomous Systems. Vol.55, Num. 12, pp. 841-850. DOI:10.1016/j.robot.2007.07.010

[11] J.M.Cañas, J. Ruíz-Ayúcar, C. Agüero, F. Martín. *JDE-neoc: component oriented software architecture for robotics*. Journal of Physical Agents, Volume 1, Number 1, pp 1-6, 2007.

[12] Francisco Martín , Vicente Matellán, José María Cañas y Pablo Barrera. *Localization of legged robots based on fuzzy logic and a population of extended kalman filters*. Robotics and Autonomous Systems. Vol.55, Num. 12, pp. 870-880.doi:10.1016/j.robot.2007.09.006

[13] Renato Samperio, Housheng Hu, Francisco Martín , Vicente Matellán. *A hybrid approach to fast and accurate localisation for legged robots* . Robotica International, Cambridge Journals. Vol. 26, Num. 06, pp. 817-830. DOI:10.1017/S0263574708004414.

[14] David Herrero-Pérez, Humberto Martínez-Barberá, Alessandro Saffiotti, *Fuzzy self-localization using natural features in the four-legged league*, in: Lecture Notes in Computer Science, pp. 110-121. vol. 3276, 2005,

[15] J.M.Cañas, E. Perdices, T. González, D. Puig. *Recognition of Standard Platform RoboCup Goals*. Journal of Physical Agents. Volume 4, Number 1, pp 11-18, 2010. ISSN: 1888-0258

[16] J. Martínez-Gómez, J. A. Gámez, I. García-Varea and Vicente Matellán. *Using Genetic Algorithms for Real-Time Object Detection*. RoboCup 2009: Robot Soccer World Cup XIII Lecture Notes in Computer Science, 2010, Volume 5949/2010, 215-227,

[17] Juan F. García, Francisco J. Rodríguez, Vicente Matellán, Camino Fernández. *Designing a minimal reactive goalie for the RoboCup SPL* X Workshop en Agentes Físicos (WAF'2009). Cáceres (España), September 2009.

[18] Michael J. Quinlan and Richard H. Middleton, *Multiple Model Kalman Filters: A Localization Technique for RoboCup Soccer* in RoboCup 2009: Robot Soccer World Cup XIII, pp 276-287

[19] Hessel van der Molen and Arnoud Visser. Bachelor Thesis: *Self-Localization in the RoboCup Soccer Standard Platform League with the use of a Dynamic Tree*. Available: http://hesselvandermolen.dyndns.org/BachelorProject/comparisonDoc.pdf,[Aug, 23, 2011]