J Braz Comput Soc (2011) 17: 69–82
DOI 10.1007/s13173-011-0028-4

ORIGINAL PAPER

On the construction of a RoboCup small size league team

José Angelo Gurzoni Jr. · Murilo Fernandes Martins · Flavio Tonidandel · Reinaldo A.C. Bianchi

Received: 20 July 2010 / Accepted: 5 January 2011 / Published online: 26 January 2011

Abstract The Robot Soccer domain has become an important artificial intelligence test bench and a widely studied research area. It is a domain with real, dynamic, and uncertain environment, where teams of robots cooperate and face adversarial competition. To build a RoboCup Small Size League (SSL) team able to compete in the world championship requires multidisciplinary research in fields like robotic hardware development, machine learning, multi-robot systems, computer vision, control theory, and mechanics, among others.

This paper intends to provide insights about the aspects involved on the development of the RoboFEI RoboCup SSL robot soccer team and to present the contributions produced over its course. Among these contributions, a computer vision system employing an artificial neural network (ANN) to recognize colors, a heuristic algorithm to recognize partially detected objects, an implementation of the known rapidly-exploring random trees (RRT) path planning algorithm with additional rules, enabling the angle of approach of the robot to be controlled, and a layered strategy software system.

Experimental results on real robots demonstrate the high performance of the vision system and the efficiency of the RRT algorithm implementation. Some strategy functions are also experimented, with empirical results showing their effectiveness.

Keywords Robotic soccer · Computer vision · Neural networks · RRT path planning · Omnidirectional control

1 Introduction

The RoboCup—Robotic Soccer Cup—domain was proposed by several researchers [17–19] in order to provide a new long-term challenge for artificial intelligence research. Since its introduction, the robotic soccer has become of relevance because it possesses several characteristics also present in other complex real problems. Examples are robotic automation systems that can be seen as a group of robots in an assembly task [28, 37], group transport tasks [30], and space missions with multiple robots [15, 38].

Soccer games between robots, as the one depicted in Fig. 1, constitute real experimentation and testing activities for the development of intelligent, autonomous robots, which cooperate among themselves to achieve a goal.

The development of a robot soccer team involves a wide range of technologies, including the design of autonomous agents, multi-agent collaboration, strategy definition, real-time reasoning, robotics, machine learning, control theory, computer vision, and sensor-fusion. This work provides an overview on how these technologies are grouped, and together form a team of robots capable of playing soccer in the RoboCup Small Size League.

Murilo Fernandes Martins thanks the support of CAPES through the grant 5031/06-0.

J.A. Gurzoni Jr. (✉) · F. Tonidandel · R.A.C. Bianchi
Department of Electrical Engineering and Computer Science, Centro Universitario da FEI, Av. Humberto de Alencar Castelo Branco, 3972, 09850-901, Sao Bernardo do Campo, SP, Brazil
e-mail: jgurzoni@ieee.org

F. Tonidandel
e-mail: flaviot@fei.edu.br

R.A.C. Bianchi
e-mail: rbianchi@fei.edu.br

M.F. Martins
Department of Electrical and Electronic Engineering, Imperial College London, London, UK
e-mail: mfmartin@imperial.ac.uk

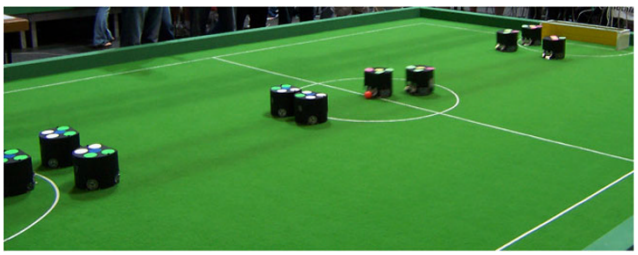


Fig. 1 Small Size League match (source: RoboCup Federation)

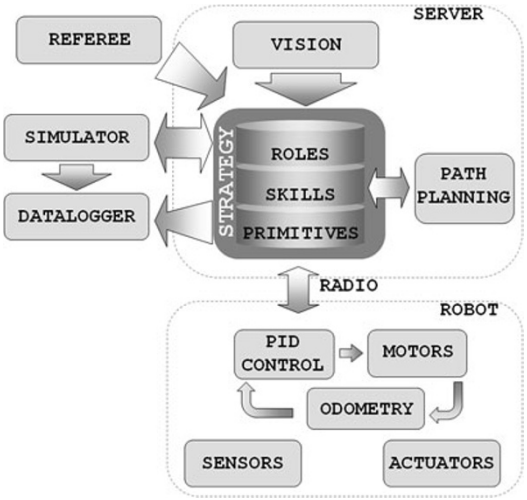


Fig. 2 Block diagram of the system

The paper is organized as follows: Section 2 provides an overview of the Small Size League and shows the tasks involved on the creation of a competitive team for this league. Subsequent sections describe key aspects of the research, namely the vision system (Sect. 3), the robotic motion control (Sect. 4), the path planner, and obstacle avoidance (Sect. 5) and the strategy (Sect. 6). Section 7 presents the experiments performed and shows the results obtained, while the conclusions are presented in Sect. 8.

2 Overview

There are several topics to consider and disciplines required into the task of building a RoboCup Small Size League [31] (also called SSL for short, or F180, due to the diameter of the robots) team able to compete in the world championship.

In the SSL, teams of 5 robots, sized up to 180 mm diameter and 150 mm height, compete in a play field of 6.1×4.0 m, using global vision systems with cameras positioned over the field, at 4 m height. The human referee uses a referee software to send game state information directly to the computers controlling the teams, via IP network. No other human interaction with the robots or with their controller computers is allowed, while the game is running. As there are very few restrictions about what sensors, actuators, and communication devices a robot can have, many possibilities exist, such as the use of kicking, dribbling, and ball lifting devices. In fact, without at least these three devices, chances of success greatly decay.

A SSL game is also very dynamic, requiring robust hardware and advanced software and sensory, because several teams have robots capable of running at speeds above 5 m/s and kicking the ball at 10 m/s speed. These requirements led to a number of solutions developed, therefore, we recommend the readers interested by the topic of this paper to also read the papers describing the CMDragons [11], the B-Smart [22], and the Skuba [35] teams.

The main topics involved in building a team for the Small Size League are: (i) the assembly of a robust electronics circuitry and mechanical platform (not covered on this paper), (ii) control engineering for the robot motion control, (iii) the creation of a computer vision system, (iv) a path planning and obstacle avoidance (sometimes also called navigation) system, and finally, (v) a strategy system, which is usually the main interest of researchers working on robot soccer, and a portion of the work that will only function provided all the previously listed topics are operating properly.

Figure 2 shows the main modules of the RoboFEI SSL team architecture. Vision, path planning, strategy, and robotic motion control modules will be detailed in their specific sections, along this paper. The simulator and data logger are discussed in this section.

The simulator engine is an important part of the software tools needed for developing a SSL team. It must offer at least strategy visualization/simulation and contain basic physics simulation. Although there are existing simulators [6, 23] featuring advanced physics engines, like ODE, in this research, the Teambots [3, 4, 20], a Java simulator with basic mechanics, was chosen as simulation engine. The last of its software releases dates of year 2000, so its simulated world and teams had to be modified, updating the modules to the current game rules, to the omnidirectional robotic hardware and to include the electronic referee commands.

Having data logging capabilities is also important, to record the information obtained from sensors, the responses given, and decisions made by the software modules. Both the simulator, the data logger, and the referee can be connected to the strategy modules via network sockets, shared memory, pipes, or any other method that provides communication, as long as the method is reliable.

Finally, a recommendation is to make the software architecture as much modular as possible, segregating the main modules in a manner that allows their independent development and execution.

3 Vision system

Some researchers claim that computer vision, responsible for the recognition of the robots and ball pose, is the richest perceptual data of a robot soccer system. The computer vision system must retrieve the pose of the objects recognized in the scene accurately, in real time, and should cope with luminance variation, occlusion, lens distortion, and noise. Also, as different teams and robots are distinguished by different colors and shapes, the computer vision system must recognize these colors and shapes in order to provide the right information about the scene observed. The only rule imposed in the SSL for vision systems is the restriction that each team must have a color marker with a predefined area, either yellow or blue, and the opponent team must not use that same color.

The problem is frequently dealt with in two levels: at the lower, there is a color and/or shape segmentation algorithm, while at the higher level there is a classification algorithm, responsible for receiving the lower level results and reasoning about it, discarding the unavoidable noise and dealing with eventual uncertainties.

For the lower level system, as stated earlier, the obvious choices are segmentation based on color, shape, or an hybrid of the two. Color segmentation is still an open issue and worthy of consideration, as recent studies have shown (e.g., [34, 39]), due to difficulties in dealing effectively with luminance variation and color calibration. To avoid pure color segmentation, in previous work [26] by some of the authors, a shape-based system that did not use color information for object segmentation, only for object distinction, was created. As a result of this shape-based approach, the system proved to be fast and more robust to luminance variation. Basically, the system consisted of subtracting the static background of the image, leaving only foreground, then finding circles in the image and color classifying the pixels inside these circles. However, color calibration still required expertise and had limitations when under uneven luminance conditions. To solve the color classification issue, in a subsequent work by the authors [14], an Artificial Neural Network (ANN) was introduced to classify the colors automatically and increase the accuracy. The result was a system with fast segmentation and accurate color classification, with the shape-based algorithm finding the circles and eliminating most of the noise, while the neural network could recognize colors even under changing or uneven luminance.

However, the vision system described on the previous paragraphs, that served very well for the IEEE Very Small League, failed in the RoboCup SSL, where the camera is posted at 4 m height and color markers of 5 cm^2 have to be found on the field. Distortions caused by the high speed of the robots and ball, besides the size of only a few pixels of the objects, rendered the shape-based segmentation inviable. Under these conditions, circles became elliptical objects at best, more often not even forming defined shapes. High speed, high resolution cameras helped, but did not solve the problem.

A new vision system had to be introduced, with the lower level segmentation algorithm that could cope with sudden shape changes, whilst keeping computational costs low. This new vision system is based on the CMVision system [10], based in color classification and blob coloring creation. The differences to the CMVision are the use of an artificial neural network color classifier and a heuristic algorithm to resolve uncertainties in the object detection.

The coming subsections describe the key aspects of the solution: the artificial neural network used to perform the color segmentation, the simple method to overcome its high computational cost, the run length encoding algorithm employed to create the blob objects, and the heuristic based algorithm that selects best candidates, among the list of potential objects, and provides tracking capabilities.

Experimental results of the neural network algorithm for color recognition will be presented in Sect. 7.1.

3.1 Artificial neural networks to recognize colors

Artificial Neural Networks, particularly Multilayer Perceptron (MLP) networks, have been applied to a wide range of problems such as pattern recognition and robot control strategies, noticeably for their capability of approximating non-linear functions, by learning from input-output example pairs, and generalization (see [29]). MLP neural networks can learn from complex and noisy training sets without the need to explicitly define the function to be approximated, which is usually multi-variable, non-linear, or unknown. In addition, they also have the capability to generalize, retrieving correct outputs even when the input data is not present in the training set. Generalization also allows the network to be trained with a small quantity of samples.

One aspect that makes neural networks attractive for use in the color classification problem is their ability to represent non-linear, discontinuous functions. Because of luminance variations, the representation of a given color class, be it in the RGB or HSV domains (see [13] for a description of these color domains), often occupies a variable number of discontinuous portions of the space. Such discontinuities cannot be represented in a manually-defined thresholding system, unless it has one set of thresholds for each of these discontinuous portions. Nevertheless, creating multiple thresholds increases considerably the expertise and time required to calibrate the colors, should the task be manually performed by the operator. With neural networks, these discontinuous representations can be quickly learned from the training set, without the need of operator's expertise, making them a more suitable solution for the task.

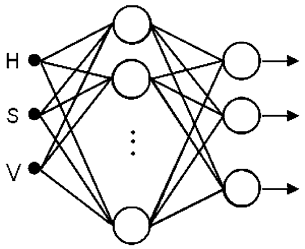


Fig. 3 The MLP neural network implemented to recognize colors

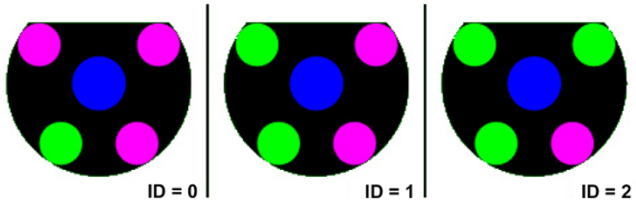


Fig. 4 Three examples of the Butterfly pattern style

Similar to [33], the MLP neural network employed in the vision system, shown in Fig. 3, is composed of three layers. The hidden and output layers consist of neurons activated by sigmoid functions, and training is performed using the well-known back-propagation algorithm. The input layer has three inputs, same dimension of the HSV space, while the output layer has seven computing units: six outputs represent each of the colors of interest (orange, yellow, blue, green, pink, and cyan) and one output represents the background. Regarding the hidden layer size, after some iterative trials, the best accuracy was found to be around 20 neurons. As a result, the topology of the MLP neural network was set as 3 inputs, 20 neurons in the hidden layer, and 7 neurons in the output.

Neural networks, however, have a relatively high computational cost, due to the fact that each new input sample requires a new calculation of the neuron matrices, which cannot be computed a priori. Fortunately, to overcome this limitation, a very simple technique can be applied: Instead of querying the neural network during the game, a tridimensional lookup table is created and preloaded with the output of the neural network to all the possible values of the HSV space ($360 \times 100 \times 100$ memory positions).

3.2 Blob creation using the RLE algorithm

Once the image is completely classified with the colors of interest, a process needs to transform the individually classified pixels into connected regions that can be treated as objects, the so-called blobs, because they do not have a known shape. This is potentially an expensive operation that can affect real-time performance. The CMVision [10] is one of several existing libraries that solves this problem with good performance. In fact, it could be used as a full vision system, but the course adopted in this work was to use only portions of it, merging them with other blocks, to achieve superior performance. Full details about the CMVision can be found in [10], therefore, this paper only details CMVision's Run Length Encoding (RLE) algorithm and the tree-based structure used into it. Run length encoding is a simple compression algorithm, in which long runs of data (long sequences of the same data value) are stored as a single data value and count. In this specific case, the sequences are the segmentation values of each pixel (computed using the aforementioned neural network). The blob creation algorithm applies the RLE on each row of the segmented image, so at the end of its execution, the result is an image in which pixels are grouped in rows of connected regions. Then it groups connected regions of same data value belonging to adjacent rows, scanning the image vertically. CMVision accomplishes these region connections among rows efficiently, because the regions are stored in a tree-based data structure, allowing an union-find algorithm to group them quickly.

At the end of the algorithm, when it has finished with the region connections (blobs), it extracts the region information such as bounding box, centroid, and area, necessary for the analysis of the pattern recognition and heuristic algorithms.

3.3 Vision reasoning algorithm

Although the vision system has already detected all the blobs and has information about their color, position, and size, the object is still not detected, just its components. At this point, the pattern detection phase, responsible for actually identifying each unique object, starts. The algorithm is designed to search for groups of blobs that form predefined patterns, unique to each robot in the field, such as the one depicted in Fig. 4, the butterfly [9].

The butterfly is a pattern that became very popular among the SSL teams. It consists of 5 circular patches: the central indicates to what team (yellow or blue) the robot belongs. The other four are used to identify each of the twelve unique combinations, using only 2 different colors. To identify the orientation of the robot on the plane, the butterfly pattern uses different distances between the two patches on the front and the two patches on the rear of the robot.

However, there may still exist reasonable amounts of noise and uncertainty in the image, resulting from noise itself, partial occlusions, errors in the segmentation and classification processes, or camera pose discrepancies. For this reason, the pattern matching has been enhanced with heuristics, like the minimum change concept and a best matching algorithm.

The minimum change concept states that robots and ball displacements are constrained by their physics and the vision samples at a fixed interval, so it shall be a reasonable assumption that, in case of multiple candidates for a given

object, the closer the candidate is to the object's previous location the more likely it is to be the correct one. This heuristic is modeled as shown in (1), a uni-dimensional Gaussian kernel. The kernel is centered on the previous location of the object, with variance given by a cache of the last second of the object's displacement data.

$$h = \frac{1}{\sqrt{2\pi}\sigma_d} \exp\left(-\frac{(d-\mu_d)^2}{2\sigma_d^2}\right) \tag{1}$$

where:

- h is the normalized Gaussian kernel value;
- d is the displacement between the current and last detected positions of the object;
- μ_d is the average displacement measures;
- σ_d and σ_d^2 are the standard deviation and variance, respectively.

The best matching algorithm, set for the pattern type used, is then executed, returning the likelihood the groups of blobs found have to be objects. The best matching algorithm mimics the pattern detection operation, but seeking for other groups of blobs that might form robots. An example is a butterfly style object where one of the circles was not detected. In this case, the algorithm would detect neighboring circles that do not form any known pattern, but match the distance parameters among themselves. These neighboring circles have a non-zero probability, in the form (object | blobsfound), of representing an object. In the current algorithm implementation, the probability values of incomplete detections actually belonging to objects were manually entered a priori. Automatically adjusting these values though would be desirable.

The two heuristics shown are weighted summed to produce the final likelihood that groups of blobs have for being considered objects. Then the groups with larger sum results are selected as objects. Presently, the weights are empirically chosen, but their automation is planned for the near future.

4 Omnidirectional drive control

Most teams competing on the SSL League today [16, 40] use robots designed using omnidirectional drive. This drive system allows the robot to translate between any two points on the plane without having to rotate its body toward the goal direction first, e.g., like a car. To achieve this, special wheels are employed, assembled with smaller wheels attached to the periphery of the main one. These smaller wheels are assembled in a position that allows them to rotate frictionless in a direction other than the main wheel, connected to the motor axis. There are a number of different omnidirectional wheels, like the Swedish or Mecanum wheels, not covered in this section, as they are unusual in Small Size robots. The wheel type usually chosen for is the one where the small wheels are mounted with the rotation axis perpendicular to the main wheel's axis. A mechanical design of the robots, showing the omnidirectional wheels and their disposition, is shown in Fig. 5.

Any robot with three or more wheels can be designed as omnidirectional drive, however, most of the robots built to compete on the SSL have four. There are two main reasons favoring the use of four and not three wheels: (i) more motors mean more torque, therefore, more acceleration, what is highly desirable on a game where robots can cross the field in little more than one second. (ii) With three wheels, movement to certain directions has significantly less speed and acceleration than others, what leads to more constraints when positioning the wheels, to make sure the robot drives fast enough at least when driving forwards. These constraints become even worse to deal when one recalls that wheels are not the only parts assembled on the periphery of the robot. The kicking, dribbling, and passing devices also need to be positioned. Four wheels alleviate these constraints.

Once the number of wheels is chosen, their mounting position needs to be defined, so the movement matrices can be calculated. For a more in-depth understanding about the construction and control of omnidirectional drive robots, the reading of [32] is recommended. In this section, the main equations and a methodology to control the robot's movement are presented.

When building an omnidirectional control, the goal is to control the robot's translational speeds on the plane, v_x, v_y and the rotational (angular) speed w, while having as actuators the motor forces f_1, f_2, f_3, and f_4. To transform forces into motor speeds, first the robot must be described in terms of its force matrix C, velocity coupling matrix D, mass M, mass distribution α, and radius R. The force and velocity coupling matrices depend on the wheels disposition, as their respective equations, (2) and (3), show. The angles φ_n of these matrices are shown in Fig. 5(right) and are measured in relation to the horizontal axis of their quadrants.

$$(a_x, a_y, R\dot{w})^{\mathrm{T}} = \frac{1}{M} C(\mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3, \mathbf{f}_4)^{\mathrm{T}} \tag{2}$$

$$(\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \mathbf{v}_4)^{\mathrm{T}} = D(v_x, v_y, Rw)^{\mathrm{T}} \tag{3}$$

where:

$$C = \begin{pmatrix} -\sin\varphi_1 & -\sin\varphi_2 & \sin\varphi_3 & \sin\varphi_4 \\ \cos\varphi_1 & -\cos\varphi_2 & -\cos\varphi_3 & \cos\varphi_4 \\ \frac{1}{\alpha} & \frac{1}{\alpha} & \frac{1}{\alpha} & \frac{1}{\alpha} \end{pmatrix}$$

$$D = \begin{pmatrix} -\sin\varphi_1 & \cos\varphi_1 & 1 \\ -\sin\varphi_2 & -\cos\varphi_2 & 1 \\ \sin\varphi_3 & -\cos\varphi_3 & 1 \\ \sin\varphi_4 & \cos\varphi_4 & 1 \end{pmatrix}$$

Having all the terms, the matrices C and D are assembled, and control loops can be used to control the translation

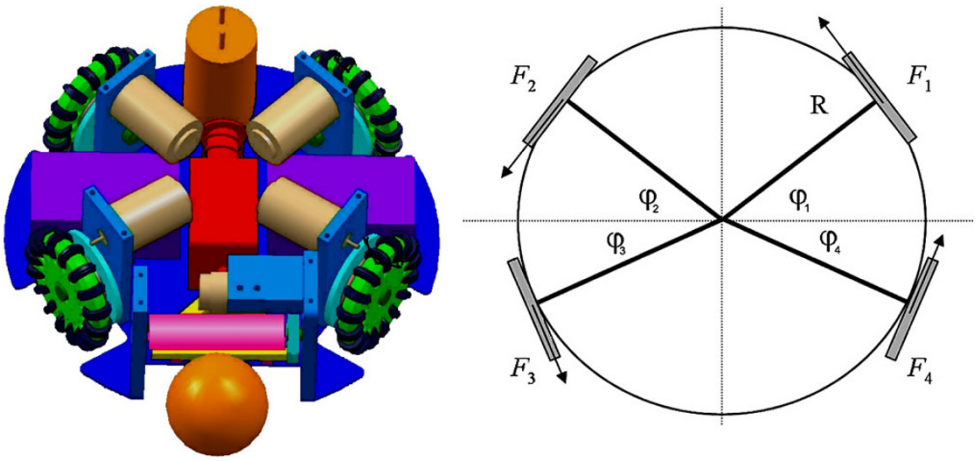


Fig. 5 (*Left*) Mechanical view of the robot's base. (*Right*) Wheel disposition diagram

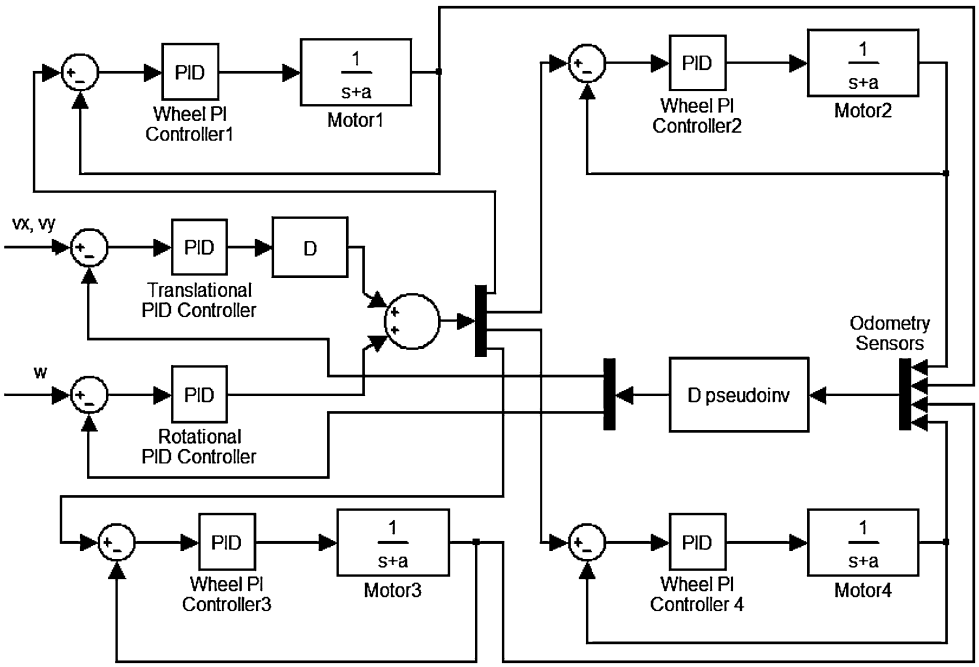


Fig. 6 Block diagram of the control loops embedded into the robots

and rotation of the robot. The feedback for these loops can be odometry measures, when the control loops are embedded into the robot, or the vision system, when the control is executed on the remote computer. To the knowledge of the authors, no team tried to transmit odometry information back to the computer via radio. Some very successful teams in SSL, such as the Skuba team [35], use vision as feedback, and force-torque controllers for the wheels. Others embed the control into the robot, to avoid having to deal with the latency of the vision loop.

The control system implemented in the RoboFEI team, shown in Fig. 6, is of the embedded type, and uses wheel odometry as feedback sensors. Individual PI controllers are present in each wheel, responsible for making them to rotate at the commanded speed, and interleaved with them, are two PID controllers responsible for the robot motion. One of these controllers is the translational controller, responsible for the translating the robot on the plane, having as input the velocities v_x and v_y and the accelerations a_x and a_y. The other controller is the rotational PID controller, which controls the rotation of the robot on its axis, having as input w. Both controllers use the same (3) to produce the output speeds of each wheel, v_1, v_2, v_3, and v_4, but each of them operate on different terms. Because the equation can be split in sums of translational (v_x and v_y) and rotational (w) terms, it is possible to use separate controllers, with different parameters, for these two portions of the movement.

The odometry information collected from the wheels is fed into the pseudo-inverse (non-square matrices do not have inverses) of the D matrix, and the output is then used as feedback for the translational and rotational controllers.

5 Path planning and obstacle avoidance

Path planning is one of the challenging subjects in most mobile robotic applications. As such, many different solutions were developed and applied with good practical results, but none being yet considered to be the optimal solution. Path planning can be, in a simplistic manner, reduced to the problem of how to find an *admissible* path from a starting to an ending point on the space, with admissible meaning (i) a collision-free path between start and end, as well as (ii) a path the robot can execute, given its kinematics constraints. Solutions for the first of these two items were largely studied over the last two or three decades, with solutions ranging from behavioral systems [1] and potential fields to computer geometry methods, like Cell Decomposition and RoadMap methods (i.e., Voronoi diagrams, Visibility graphs and related). Latombe [21] provides a good exposition of these aforementioned techniques applied to robotic motion. More recently, methods that employ sampling algorithms became popular, such as the Probabilistic RoadMaps (PRMs) [25] and the Rapidly-Exploring Random Trees (RRTs) [8, 12].

Solutions to the second part of the problem, whether the robot can actually accomplish the generated path, started with the utilization of the robot's kinematic model during the generation of the path, to restrict the path segments created, and then evolved to solutions that consider both the dynamics and the kinematics of the robots.

Regarding the ability to accomplish the generated path, having omnidirectional drive robots presents an advantage: the capability to move to any direction without having to rotate its front, and with nearly the same speed, allowing much less restrictions on the trajectories the robot can execute. Therefore, given the robot can execute most of the paths traced, the issue becomes to efficiently generate a collision-free path. The remaining part of this section presents a path planner that is capable of such.

The implemented path planner has a RRT [24] in its base, due specially to (i) its capacity to efficiently explore large state spaces using randomization and the probabilistic completeness offered (ii) the lookahead feature of the algorithm and (iii) its easy extension if new constraints or heuristics are deemed necessary.

The efficient exploration of a large state space via randomization provides a sound guarantee of the *theoretical* probabilistic completeness, the assurance of a non zero probability of finding a path if one exists. Must be noticed though that the term *theoretical* is left in evidence because this statement only holds true if the tree is allowed to grow for infinite iterations. In practice, however, the number of cases it could not find a path was very reduced. The lookahead feature has a considerable advantage over reactive based systems such as the potential fields, helping eliminate most of the sub-optimal paths and local minima, because the tree generated finds a path between start and end is the vast majority of the cases. And last, the easiness of the addition of constraints or heuristics comes from the incremental nature of the algorithm. One can add more rules to allow or not the expansion of the tree to occur toward a certain state or region of the space without having to disassemble the algorithm logic.

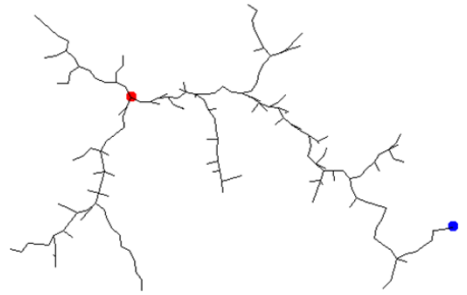
5.1 KD-trees as RRT structures

The implementation here described is based on the Rapidly-Exploring Random Tree (RRT) algorithm proposed on [2]. The aforementioned study states that, because RRTs grow by selecting a new point on the plane and expanding toward it from the nearest existing node, at each iteration, they are heavily dependent on nearest neighbor searching. Therefore, the usage of a data structure capable of improving this search would improve the whole RRT algorithm significantly. KD-Trees (short for *k-dimensional* trees) are data structures created exactly to optimize nearest neighbor searching in k-dimensional orthogonal planes, by recursively subdividing a set of points based on alternating axis-aligned hyperplanes. In short, it can be said the KD-tree sorts the set of k-dimensional points in all its dimensions by sorting each level of the tree in one axis. The classical KD-tree uses $O(d \log n)$ construction time, and answers queries[1] in $O(n^{1-1/d})$, where n is the number of k-dimensional nodes in the tree and d the dimension. There are more advanced versions of the nearest neighbor algorithm using KD-Trees, such as the approximated nearest neighbor search, which could be utilized in future implementations, but as of now they are not used.

5.2 Implementation details

As stated in Sect. 5.1, the algorithm is based on the RRT with KD-Trees from [2], the ERRT algorithm developed by [8] (also a comprehensive reference for those seeking a RRT path planner implementation guide), which adds concepts such as the straight line segment replacement and the planner 'memory' concept with the use of a cache of the previously found path, called *waypoint* cache and algorithms to include preferred path heuristics and set the angle of approach. Let p be the probability of the RRT tree to be expanded toward the goal, and q be the probability of the tree being expanded to a point belonging to the previous successfully generated path (this probability is suppressed until the first successful path is generated). Let also l be the maximum expansion length, which effectively gives the maximum size of the tree's path segments. The shorter this length l is, the more precision the path would have, but it would

[1] For orthogonal range queries.



(a) 20%

(b) 35%

(c) 50%

Fig. 7 RRT tree examples with different probabilities toward goal, without waypoint cache. (**a**) has 20% probability, (**b**) has 35%, and (**c**) has 50%

also take more steps to reach the goal. Figure 7 shows examples of different probabilities p set.

Having defined these values, the initial RRT tree (stored in a KD-Tree) is created, containing only the starting point, then extended iteratively (if obstacle avoidance rules described next allow) as follows: With probability p the tree is extended toward the goal; With probability q it is expanded toward a point from the previous path, stored in the waypoint cache; With probability $1 - p - q$ it is expanded toward a randomly selected point on the plane. All the expansion lengths are limited by the length value l. The iterations end when the goal point is reached or when the number of iterations has exceeded its maximum, a limitation needed to bound the execution time in case of uncommon situations such as a collision-free path being inexistent. Once finished, as described in [8], the path is replaced by the minimum number of straight lines that can connect the start and end points without intersecting obstacles. Also, at this point, the waypoint cache is updated with a constant number of points (the implementation described uses between 50 and 100) taken from the path found. This cache array will be used as memory of the algorithm on the next operation cycle with q probability of being selected, thus avoiding oscillations

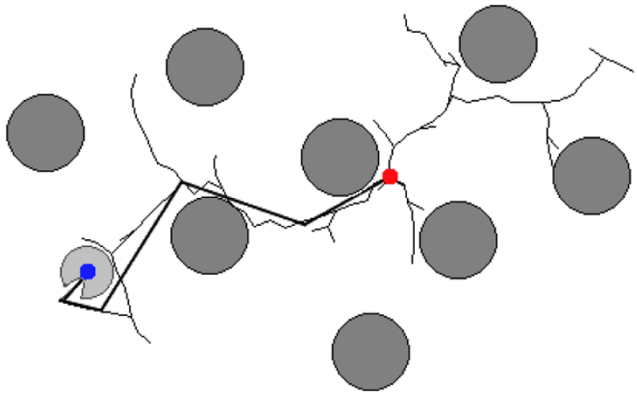


Fig. 8 Example of the full implementation of our path planning algorithm. The *line segments* shows the RRT and *circles* are obstacles. The *circle segment around the ending point* is the angle of approach constraint. In *bold*, the final chosen path, starting from the point on the right

between different paths that might arise due to the random nature of the search and reducing the number of nodes expanded.

Obstacle avoidance and other constraints are implemented in a simple manner: During the tree expansion phase described in the previous paragraph, after expanding the tree toward a given point and before to actually store it in the tree, the new node is tested against a rule or set of rules to determine if the tree should be expanded or the expansion canceled. This is called the expansion metric of the tree.

The current implementation uses three sets of metrics as expansion metric of the tree. The first rule checks if the node intersects with obstacles. Each robot is set, regardless of teammate or opponent, as a circular obstacle slightly bigger than its actual size. If an intersection of the node with the area where an obstacle lies is found, the node is discarded from the tree. If not, it is kept. The flexible nature of the algorithm allows not only the robots to be set as obstacles but also the goal areas, as these are positions forbidden by the game rules, or any other region of the field desired.

The second rule exists to set the angle which the robot approaches the ending point to the desired by the strategy layer, an item that many path planners do not treat. It is not desirable, for example, that a robot going to the ball on the defensive field accidentally hits the ball in the direction of its own goal, or yet that an attacking robot arrives at the ball in a position in between the ball and the opponent's goal. To create a path that conforms to the angle of approach requirement, a circular virtual obstacle centered on the ending point is created, with a 20° width circle segment and vertex at the desired angle removed. This effectively forces the path planner to create a path that reaches the ending point passing through this 20° opening. The radius of this obstacle-like constraint is set at around half of the size of a robot. Figure 8 shows an example of the algorithm's result.

On top of these, a heuristic map is also created. This map, controlled by the strategy layer, can modify the probability given areas of the field have of being randomly selected for tree expansion, effectively biasing the robots to avoid or favor portions of the field, when creating paths. More details on the heuristic map can be found in Sect. 6.

6 Strategy

Building multi-agent systems in a layered architecture with different levels of abstraction is a popular approach (see [27]) in complex systems. Examples of this kind of layered architecture with good results can also be found in SSL teams, such as [5, 36], and the STP [7], which is likely the most famous of these architectures.

The STP divides the strategy system in three layers: skills, tactics and plays. Skills are short term actions that the robots can do individually, like shooting to goal or passing. Tactics are state machines that encapsulate long term behaviors that the robot must execute, such as going to the attack field, receiving the ball from the teammate and shooting to the goal. Plays are collective state machines, involving a number of robots. The limitation of the STP, according to its authors, lie in the difficulty to change existing or add new tactics, and plays, because the criteria used to select between state machines and the hard coded states within them are very tightly coupled.

The strategy architecture presented in this paper, like the STP, is composed of three layers, named Primitives, Skills, and Roles. The Primitives and Skills are similar to the Skills and Tactics of the STP, respectively. The difference is in the Roles layer, which allows each robot to decide by himself, without the need for collective state machines.

The lowest layer, the *Primitives* layer, is composed by actions that mostly involve directly activating or deactivating a hardware module such as to kick the ball with a given strength, activate the dribbling device, rotate or move to a position.[2]

On top of the primitive layer comes the *Skills* layer. Skills are also short duration actions but involving use of one or more primitives and additional computation, such as angle calculations, speed estimation or forecasting of objects' positions, measurement of a primitive task's completion, and verification of obstacles displacement. On this layer, as few skills as possible where created, while still maintaining a set of skills that would represent all the basic skills required in a robot soccer game like shooting the ball to the goal (aiming where to shoot), passing the ball to a teammate rolling it on the floor or lifting it on the air, dribbling, defending the goal line, or tackling the ball (moving toward the ball and kicking it away). The skills as well as the roles which group them are shown in Table 1. Before entering the description of the role layer, the structure of a skill is presented, showing in more details an offensive skill, the shoot to goal algorithm, and a defensive skill, the Delta defense.

Table 1 Roles and its Skill sets

Role	Skill Set
Attacker	DribleToPosition(x, y)
	ShootToGoal()
	PassTo_floor()
	PassTo_lift()
	ReceivePass()
Defender	DeltaDefense()
	TackleBall()
	PassTo_floor()
	PassTo_lift()
MidFielder	DribleToPosition(x, y)
	ShootToGoal()
	TackleBall()
	PassTo_floor()
	PassTo_lift()
	ReceivePass()
GoalKeeper	DefendGoal()
	PassTo_floor()
	PassTo_lift()

ShootToGoal() has the purpose of selecting the best angle to shoot to the goal, evaluating obstacles and opponent's goalkeeper positions, aligning the robot to that angle, and activating its kicking device. The function core is the aiming algorithm, shown in Algorithm 1. The aiming algorithm starts by defining a triangle between the ball and the goal limits and retrieving the angle of the ball's vertex. Then it partitions this triangle in smaller triangles, also with vertex on the ball, using the line segments with a point on the vertex and tangential to an obstacle, marking also if that angle is composed by the upper or lower (in relation to the x axis) line tangential to the circle of the obstacle. Angles of lines above are classified as *blockedStart*, while the ones below are called *blockedEnd*. The next step is to sort the vector A of these angles and scan it, as follows: every angle range between a blockedEnd and a blockedStart type is selected, as these are angle ranges where there is no obstacle. The widest of these ranges is selected then, as the best angle. If there is no range available, the robot randomly selects either the left or rightmost side of the goal and kicks, aiming at that side.

DeltaDefense(), a skill with the objective of forming a line in front of the goalkeeper, to help protect the goal. It

[2] Actually, moving to a position is a special case of a primitive with underlying complex logic. It calls the path planning system to perform obstacle avoidance.

is named *Delta* defense because it forms a triangle between the goalkeeper and the two defenders, when both of them execute the function at same time. The defender positions itself just in front of the defense area, slightly to the left or slightly to the right in relation to where the goalkeeper is, while also looking for the other defender, to make sure they are not trying to occupy the same position.

Once the skills are in place, they are employed by the *Roles* layer, that creates different roles using combinations of skills and logic to coordinate the execution of these skills accordingly. The current implementation has four types of roles, called goalkeeper, defender, midfielder, and attacker.

To exemplify how roles can make different uses of skills, consider the behaviors of the attacker and defender roles in relation to the skill *passTo_floor()*, a function that passes the ball rolling on the floor to the teammate with better chance to carry it or to shoot. The logic of the attacker is to create a probability gridmap of the field area between itself and the opponent's goal and to fill the probabilities of this gridmap. The function used to determine the values of these probabilities calculates a weighted sum of the chances to fail in a shoot to goal, based on angle, proximity, and obstacles. The attacker's logic also continuously searches for the best teammate to receive the ball, based on their proximity to opponents, angle to kick to goal and angle between the ball and the goal. On the other hand, the defender role's gridmap is filled with the weighted sum of the chances to successfully pass the ball to a teammate, favoring the closest to the opponent's goal, and the probability of being tackled by an opponent.

There are major advantages in segmenting the strategy system into layers, such as the capability to use simple actions in lower level layers as building blocks for the higher level actions, and to use these building blocks into many higher level functions, what effectively makes it a modular system.

Results of the application of this layered strategy, acquired in game scenarios, can be seen in the experiments and results section (Sect. 7.3).

Algorithm 1 AimToGoal()

Require: p and ϕ—structs with two angles; A—a vector with angles and their types.

$\phi \leftarrow$ Get Angle Range between ball and the goal limits
for each robot not behind x coordinate of ball **do**
  $p \leftarrow GetAngleBlocked(ball.coords, robot.coords, r)$
  if $p.blockedStart$ within ϕ **then**
    add $p.blockedStart$ to A
  end if
  if $p.end$ within ϕ **then**
    add $p.blockedEnd$ to A
  end if
end for
add $\phi.blockedStart$ to A, swapping its type
add $\phi.blockedEnd$ to A, swapping its type
Sort(A)
for $i \leftarrow 1$ to $Length(A)$ **do**
  if $A[i-1].type = blockedEnd$ and $A[i].type = blockedStart$ **then**
    add angle range from $A[i-1].blockedEnd$ to $A[i].blockedStart$ to $OpenAngles$
  end if
end for
return max($OpenAngles$)

7 Experiments and results

This section shows the experiments performed with the different modules of the system, and respective results. All tests were executed on the real robots, operating within the game environment. The modules were also used during the RoboCup 2009 competition, with results matching the ones presented ahead.

7.1 Vision experiment

The purpose of the vision algorithm experiment was to validate the contribution presented in this work, of an accurate color calibration method, capable of correctly classifying pixels even under changing luminance and trained with noisy samples. Position and orientation accuracy results are believed to be similar to the work of [10], therefore readers interested in validating positioning accuracy are recommended to follow the methodology there described.

In the vision system experiment, the neural algorithm was calibrated with 70 samples from the live camera image, 35 of these obtained at a luminance level of 1000 lux and 35 obtained at 600 lux. The training algorithm and parameters used are the same as described in Sect. 3.1. The thresholding algorithm, used for comparison purposes, was calibrated with images taken at 6 different luminance levels (varying from 400 to 1,600 lux). After both algorithms were calibrated, the robots were set to wander across the field, moving at their normal speed to points on the field randomly chosen by the computer. Every time a robot reached its destination point or stopped moving for more than 3 seconds, a new point was chosen. A counter summed the number of times each robot was correctly identified in each of the algorithms, in a cycle of 4,000 frames, and the process was repeated for 4 luminance levels. Figure 9 shows a worst case detection count of the robot detected the lesser number of times during each cycle of 4,000 frames.

The results show that the neural algorithm is extremely robust to luminance variation, as well as to noise. On the contrary, the thresholding system is highly vulnerable to

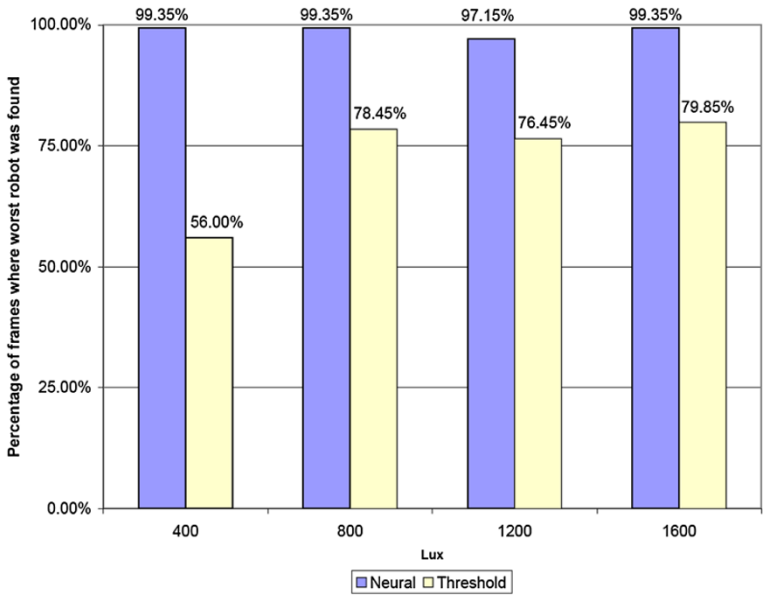


Fig. 9 Percentage of frames where the robot was correctly detected by color segmentation. Comparison between the neural network and manual thresholding methods under changing luminance conditions

noise. It is important to note also that, to obtain a calibration for the thresholding algorithm that could be used in the test, more than 45 minutes where spent in trials, for even when the results in the test images were satisfactory, after the robots started to move the results were found to be poor. The neural network calibration did not take more than 3 minutes and had to be executed only one time.

7.2 Path planner experiments

To evaluate the performance of the path planning algorithm, two test sets were performed in a real game scenario, as shown in Fig. 10, where the robot had to navigate from a starting to an ending point, passing through a portion of the field with several obstacles. In both tests, the path planner algorithm was executed at the same frequency it runs in real games, 60 times per second.

The first test was to verify the planning times during the course, while testing the algorithm in the following variations: (i) with the use of the waypoint cache, (ii) without the cache, and (iii) with the waypoint cache and using the angle of approach. The probability p toward the goal was set to 20%, while the waypoint probability q was set to 70%. The maximum length of a node was the size of half robot. The results of each of these algorithm variations, over 15 independent trials, are shown in Fig. 11. The objective of this test was to confirm the performance, as well as to observe if the waycache and the angle of approach would considerably increase the computational time required. During this test execution, the largest tree recorded had 1,564 nodes.

The second test was performed using the same scenario, but this time fixing all the parameters and changing only the probability p toward the goal (and reducing the same percentage from the waypoint probability q). The graph on

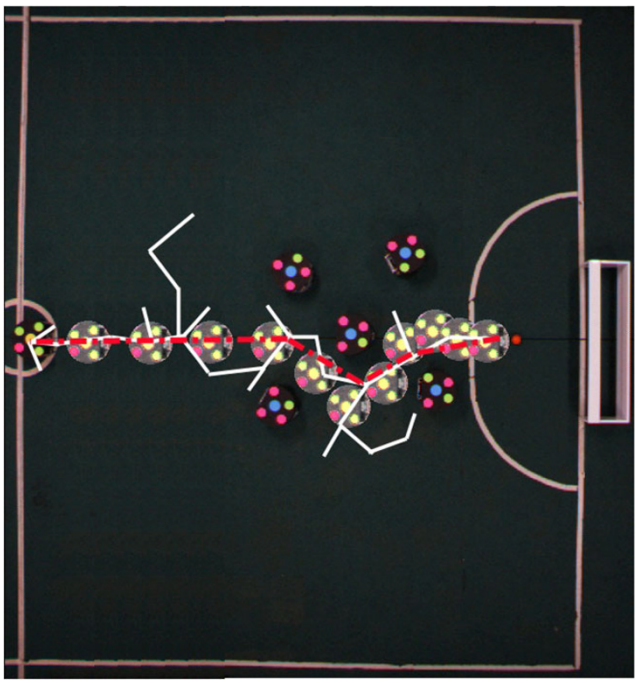


Fig. 10 Path planning test. Superimposed images show the trace of the moving robot, from its origin on the left to the goal on the right. The *white lines* show the RRT tree on the first iteration. The *red line* shows the final path given by the path planning algorithm, also on the first iteration

Fig. 12 shows a set of trials with 20% probability toward goal and 70% toward waypoint and a set of trials with 35% probability toward goal and 55% toward waypoint. The results shown are of 15 independent trials. As the probability toward the goal increases, the time spent on the planning decreases, because less exploration of the state space is performed. Test results show that there is no unique optimal

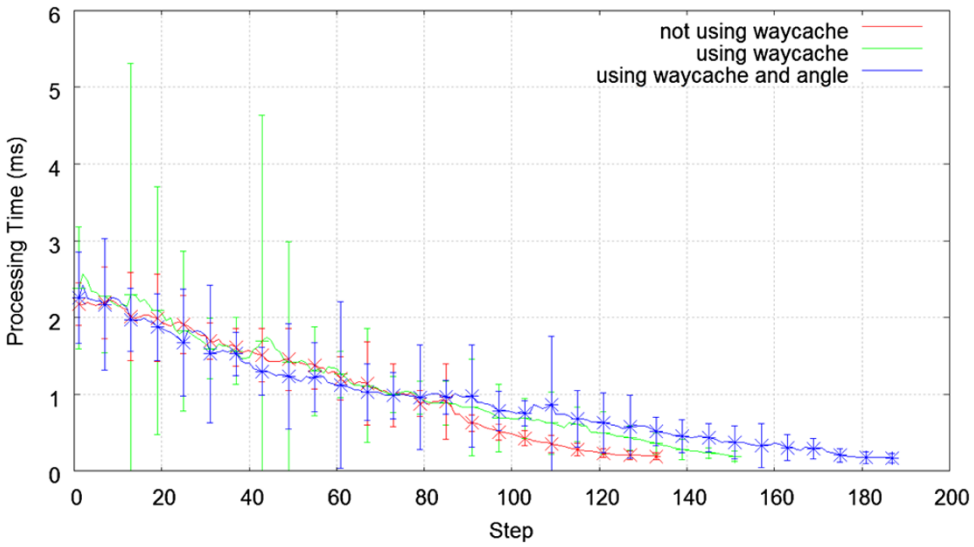


Fig. 11 Influence of the different options of the planner on the processing time, during the obstacle avoidance test. *Vertical axis* shows the algorithm processing time, while the *horizontal* shows the steps of the simulation

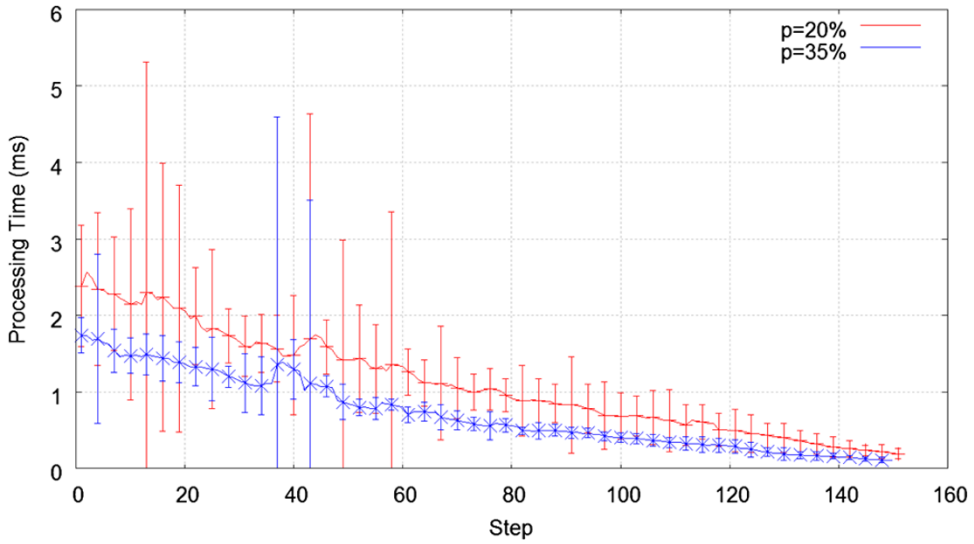


Fig. 12 Influence of different probabilities p toward the goal point on the path planner's processing time. The *vertical axis* shows time in milliseconds, while the *horizontal* shows the step of the simulation

value for the probabilities p and q, as the exploration versus smaller expansion is a trade-off dependent on how often the moving obstacles position themselves, relative to each other. An option, explored in some RRT implementations, is the use of adaptive algorithms to adjust these values, based on empirical game tests.

7.3 Strategy experiments

To test the strategy system's main offensive and defensive functions, some game scenarios were constructed. The first test consists in to randomly placing the ball in the offensive field and let the attacker shoot it to the goal, while the goalkeeper will try to defend the ball. The attacker is executing the ShootToGoal skill while the defense executes the DeltaDefense. The results of this test were compared with a scenario where the same offensive skill played against a basic defense, composed of two robots that position themselves in front of the defensive area, between the initial position of the ball and the center of the goal, and then stay put, not coordinating movements with the goalkeeper. Thirty independent runs of 100 trials each were run, and the results averaged. These results appear in Table 2, and show an improvement over the basic defense. The tests also show that algorithms such as ShootToGoal, where a single robot tries to aim and shoot, have limited effectiveness against even simple defensive schemes. Team plays, where robots pass the ball among themselves and then shoot, are good candidates to overcome this issue.

8 Conclusions and future work

The purpose of this paper was to provide an overview about the requirements and challenges involved in the creation of

Table 2 Delta defense performance

	Delta Defense	Basic Defense
Defenses	50.2%	37.0%
Balls out	34.0%	13.4%
Goals scored	15.8%	48.6%

a team capable of competing in the RoboCup SSL league. The results of the experiments demonstrate the performance of the main modules of the system, with good results in the computer vision area, where a contribution is made, a new method for color calibration based on a well-known neural network algorithm. An extension of an algorithm provenly capable for the difficult problem of mobile robots path planning is also presented, as well as a sketch of a simple AI system. In future work, we plan to extend the usage of AI techniques, especially to the layered architecture of roles, whilst building more skills, and thus expanding the role possibilities.

Acknowledgements The authors would like to thank Gabriel Francischini and Felipe Zanatto, for creating a good deal of the software, testing, debugging it thoroughly, providing insight, and feedback. The authors also thank André de Oliveira Santos, for designing all the electronics, Milton Cortez Junior, for fine tuning the mechanics of the robots, and the members of the Robotics and AI lab. of Centro Universitário da FEI, for their valuable contributions, dedication, and long hours spared. Their efforts were key to make this work possible.

We are also grateful to the anonymous reviewers for their several contributions.

References

1. Arkin RC (1987) Motor schema based navigation for a mobile robot: An approach to programming by behavior. In: Proceedings of the IEEE international conference on robotics and automation (ICRA), March, vol 4, pp 264–271
2. Atramentov A, LaValle SM (2002) Efficient nearest neighbor searching for motion planning. In: IEEE international conference on robotics and automation (ICRA), pp 632–637
3. Balch T (2000) Teambots 2.0 documentation. www.teambots.org
4. Balch T, Ram A (1998) Integrating robotic technologies with JavaBots. In: Working notes of the AAAI 1998 spring symposium, Stanford, CA.
5. Bowling M, Browning B, Veloso M (2004) Plays as effective multiagent plans enabling opponent-adaptive play selection. In: Proceedings of international conference on automated planning and scheduling (ICAPS'04), pp 376–383
6. Browning B, Tryzelaar E (2003) Ubersim: A realistic simulation engine for robot soccer. In: Proceedings of autonomous agents and multi-agent systems (AAMAS), Australia
7. Browning B, Bruce J, Bowling M, Veloso M (2005) Stp: Skills, tactics and plays for multi-robot control in adversarial environments. IEEE J Control Syst Eng 219:33–52
8. Bruce J, Veloso M (2002) Real-time randomized path planning for robot navigation. In: Proceedings of the IEEE/RSJ international conference on intelligent robots and systems (IROS)
9. Bruce J, Veloso M (2003) Fast and accurate vision-based pattern detection and identification. In: Proceedings of the IEEE international conference on robotics and automation (ICRA), Taiwan, May, pp 1277–1282
10. Bruce J, Balch T, Veloso M (2000) Fast and inexpensive color image segmentation for interactive robots. In: Proceedings of the IEEE/RSJ international conference on intelligent robots and systems (IROS), October, vol 3, pp 2061–2066
11. Bruce J, Zickler S, Licitra M, Veloso M (2008) Cmdragons: Dynamic passing and strategy on a champion robot soccer team. In: Proceedings of the IEEE international conference on robotics and automation (ICRA), Pasadena, CA, pp 4074–4079
12. Cheng P, LaValle SM (2002) Resolution complete rapidly-exploring random trees. In: IEEE international conference on robotics and automation (ICRA), pp 267–272
13. Foley JD, van Dam A, Feiner SK, Hughes JF (1995) Computer graphics: Principles and practice in C, 2nd edn. Addison-Wesley, Reading
14. Gurzoni JA Jr., Martins MF, Tonidandel F, Bianchi RAC (2009) A neural approach to real time colour recognition in the robot soccer domain. In: Proceedings of SBAI'09, The X Brazilian symposium of artificial intelligence
15. Huntsberger TL, Trebi-ollennu A, Aghazarian H, Schenker PS, Pirjanian P (2004) Distributed control of multi-robot systems engaged in tightly coupled tasks. Auton Robots 17:79–92
16. Iocchi L, Matsubara H, Weitzenfeld A, Zhou C (eds) (2009) RoboCup 2008: Robot soccer world cup XII [papers from the 12th annual RoboCup international symposium, Suzhou, China, July 15–18, 2008]. Lecture notes in computer science, vol 5399. Springer, Berlin
17. Kitano H, Asada M, Kuniyoshi Y, Noda I, Osawa E (1997) Robocup: The robot world cup initiative. In: AGENTS '97: Proceedings of the first international conference on autonomous agents. ACM, New York, pp 340–347
18. Kitano H, Asada M, Kuniyoshi Y, Noda I, Osawa E (1997) Robocup: A challenge problem for AI. AI Mag 18(1):73–85
19. Kitano H, Tambe M, Stone P, Veloso MM, Coradeschi S, Osawa E, Matsubara H, Noda I, Asada M (1997) The robocup synthetic agent challenge 97. In: International joint conference on artificial intelligence (IJCAI), pp 24–30
20. Kramer J, Scheutz M (2007) Development environments for autonomous mobile robots: A survey. Auton Robots 22(2):101–132
21. Latombe JC (1991) Robot motion planning. Kluwer Academic, Dordrecht
22. Laue T (2009) B-Smart (Bremen Small Multi-Agent Robot Team) team description for robocup 2009. In: Proceedings of the international RoboCup symposium 2009 (RoboCup 2009), June 30–July 3
23. Laue T, Spiess K, Rofer T (2006) Simrobot—a general physical robot simulator and its application in robocup. In: Bredenfeld A, Jacoff A, Noda I, Takahashi Y (eds) RoboCup 2005: Robot soccer world cup IX. Springer, Berlin, pp 173–183
24. LaValle SM (1998) Rapidly-exploring random trees: A new tool for path planning. Technical report, Iowa State University, October
25. LaValle SM, Branicky MS (2003) On the relationship between classical grid search and probabilistic roadmaps. In: Algorithmic foundations of robotics V, vol 7. Springer, Berlin, pp 59–76
26. Martins MF, Tonidandel F, Bianchi RAC (2007) Towards model-based vision systems for robot soccer teams. In: Lima P (ed) Robotic soccer, Chap. 5. I-Tech Education and Publishing, pp 95–108
27. Mataric MJ (2001) Learning in behavior-based multi-robot systems: Policies, models, and other agents. In: Cognitive systems research, April, pp 81–93

28. Mills JK, Ing JGL (1996) Dynamic modeling and control of a multi-robot system for assembly of flexible payloads with applications to automotive body assembly. J Robot Syst 13(12):817–836
29. Mitchell TM (1997) Machine learning (ISE Editions). McGraw-Hill Education, New York
30. Nouyan S, Gross R, Dorigo M, Bonani M, Mondada F (2005) Group transport along a robot chain in a self-organised robot colony. In: Proceedings of the 9th int conf on intelligent autonomous systems. IOS Press, Amsterdam, pp 433–442
31. RoboCup Federation (2010) Laws of the Small Size League 2010
32. Rojas R, Förster AG (2006) Holonomic control of a robot with an omnidirectional drive. Künstl Intell 20(2):12–17
33. Simões AS, Reali Costa AH (2000) Using neural color classification in robotic soccer domain. In: International joint conference IBERAMIA'00 Ibero-American conference on artificial intelligence and SBIA'00 Brazilian symposium on artificial intelligence, pp 208–213
34. Sridharan M, Stone P (2007) Color learning on a mobile robot: Towards full autonomy under changing illumination. In: Veloso MM (ed) International joint conference on artificial intelligence (IJCAI), pp 2212–2217
35. Srisabye J, Wasuntapichaikul P, Onman C, Sukvichai K, Damyot S, Munintarawong T, Phuangjaisri P, Tipsuwan Y (2009) Skuba 2009 extended team description. In: Proceedings of the international RoboCup symposium 2009 (RoboCup 2009), June 30–July 3.
36. Stone P, Veloso M (1999) Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork. Artif Intell 110(2):241–273
37. Sun D, Mills JK (2002) Adaptive synchronized control for coordination of multirobot assembly tasks. IEEE Trans Robot Autom 18:498–510
38. Tambe M (1998) Implementing agent teams in dynamic multi-agent environments. Appl Artif Intell 12(2–3):189–210
39. Tao W, Jin H, Zhang Y (2007) Color image segmentation based on mean shift and normalized cuts. IEEE Trans Syst Man Cybern, Part B, Cybern 37(5):1382–1389
40. Visser U, Ribeiro F, Ohashi T, Dellaert F (eds) (2008) RoboCup 2007: Robot soccer world cup XI, July 9–10, 2007, Atlanta, GA, USA. Lecture notes in computer science, vol 5001. Springer, Berlin