Submitted to WAF'2003
April 2003 (Alicante, Spain)

Vision-based schemas for an autonomous robotic soccer player

Víctor M. Gómez José M. Cañas Félix San Martín Vicente Matellán

Abstract

This article describes two vision-based behaviors designed for an autonomous robot. These behaviors have been designed and implemented using schemas according to an architecture named DSH (Dynamic Schema Hierarchies), which is also briefly described. The reactive behaviors implemented have been tested in our robotic soccer players, running in EyeBot commercial platforms. The developed behaviors provide the robot with the *follow ball*, and the *follow border* capabilities, using only the local camera and the robot on-board low processing power. The design of the control schemas, the algorithms used to analyze the images in perceptual schemas and relevant implementation issues are also described in this paper.

1 Introduction

Robot world soccer tournament, Robocup[1], is one of the current best known testbeds for physical agents[**?**]. This event aims to foster the research in the fields related to autonomous agents providing a common and attractive problem. It offers a real-time, dynamic, cooperative and competing scenario for real and simulated robots. In our group we are currently working in the Small-Size League (F-180), where robots must fit inside an 180 millimeter diameter cylinder, and where the use of a global overhead camera is allowed. In order to increase the autonomy of our robots we avoid the use of such overhead camera, and of any off board computers for our soccer players. Our main research interest focuses on architectures to get fully autonomous behaviors in robots, in such a way we would like to be more strict than current rules of F-180 category.

Two autonomous behaviors have been developed in this work to be included in our players: follow-ball and follow-border. Both are based on passive vision and they have been built and tested in the EyeBot robot[**?**] shown in figure 1. This robot is equipped with a 82x62 pixels camera, giving color images at 4 fps. It also has three infrared sensors to measure the distance to close obstacles, but they will not be used to implement these behaviors. The robot has two DC motors for the wheels, and two servos, one for camera movement an another for a kicker. All user programs run in a Motorola 68332 microprocessor, running at 35Mhz, using 1Mb. of RAM and 512KB of flash ROM memory.

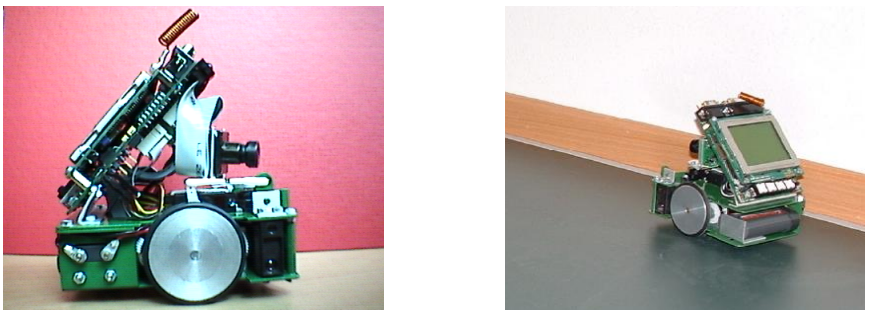

Figure 1: EyeBot robot (left) and EyeBot following a border (right)

Both follow-border and follow-ball behaviors have been designed complying an architecture named DSH[**?**]. DSH is a behavior based architecture for robot control which uses dynamic hierarchies of small schemas to generate autonomous behavior. DSH architecture is biologically inspired, using schemas from Arbib [**?**] and ideas from ethology, but takes into account engineering and programming issues to solve the main problems we are interested in, like attention and distributed action selection. Each schema is a flow of execution with a target, that can be turned on and off, and that has several parameters which tune its behavior. They may be

[1] http://www.robocup.org

independently designed as well as reused for different behaviors. Two types of schemas are defined: perceptual and motor ones. Perceptual schemas produce pieces of information that can be read by others. Motor schemas access to such information and generate their outputs, which are the motor commands or activation signal for other low level schemas (perceptual or motor) and their modulation parameters.

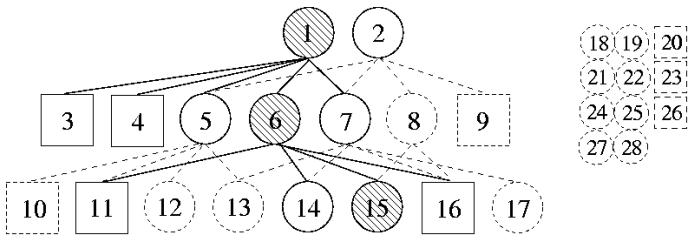

Figure 2: Schema hierarchy and pool of SLEPT schemas

Schemas are organized in hierarchies dynamically built. For instance, if a motor schema needs some information to accomplish its target then it activates relevant perceptual schemas (square boxes in figure 2) in order to collect, search, build and update such information. It may also awake a set of low level motor schemas (circles in figure 2) that implement convenient reactions to stimuli in the environment. It modulates them to behave accordingly to its own target. Low level schemas are recursively woken up and modulated by upper level schemas, forming a unique hierarchy for a given global behavior. The action selection mechanism in DSH is distributed, it sets a control competition, and chooses a winner per level, considering perpectual situation and parent goals. More details are provided in [**?**]. We will describe in the next sections follow-border and follow-wall behaviors made up by four different schemas, two perceptual, and two motor ones. No hierarchy has been needed to implement them, so we skip that architecture feature.

There are many reported works in small league from robocup, but most of them use the overhead camera [**?**][**?**] and an outer PC to take motor decisions, which are finally transmitted to the robot through a radio link. Currently there are few small robots with local camera, maybe due to the hardware complexity of putting a camera inside an small robot and providing high computing speed at the same time. Nevertheless the works using local vision in small robots are increasing in last years [**?**][**?**][**?**], most of them are adhoc designs with the aim of full autonomy. Another handicap of local vision in small robots is the typical low computing power of such platforms, so all the perceptive and control algorithms must be extremely efficient.

Braünl and Graf [**?**] employ local camera and on board processing in EyeBot soccer players. They estimate the position of the ball in the field from the local images and robot absolute location computed using encoders and a compass. Then compute a trajectory to make the robot kick the ball towards opponent's goal. Our follow ball behavior is a little bit simpler, just an active tracking problem, as no kicking is provided. Rojas et al. [**?**] use a camera directed upwards and a concave mirror to provide an omnidirectional view of robot surroundings. They use an RGB color segmetation, as our ball perception schema. In addition their color transitions technique is similar to the way we look for field edge in images for follow-border behavior. Viperoos team [**?**],[**?**] uses color segmentation in UV space instead of RGB, and they organize the architecture in different behaviors which can also be activated and deactivated, similar to DSH approach. Their behavior based architecture provides a whole repertoire of modules, including localization. The work described in this paper is just the stepping stone aiming a complete set of schemas needed for a soccer player. The hardware used in [**?**] and [**?**] provides images at 25 and 10 fps, which eases a competitive perfomance.

Remainder of the paper is organized to describe these behaviors. In the next section a follow-wall behavior based on local vision is described. This behavior will be used to drive the robot in the soccer field, which is surrounded by walls. Section 3 describes the follow-ball behavior, useful when the robot wants to kick the ball or just advance keeping it close. Some final conclusions and the future lines we envision to improve the soccer players end the paper.

2 Follow-border behavior

Current F-180 rules establish that the field of play is marked with lines and walls, which are field boundaries. So we will replace the follow-border behavior with a follow-wall behavior. The implemented behavior is prepared for a future wall supression in the rules, because it uses the floor edge as main stimuli, not the explicit wall. It has been designed as two independent schemas, which can be seen in the figure 3. The relevant stimulus

identified is the border line between the floor and the wall. The perceptive schema is in charge of identifying this stimulus in the images, and divide it into straight line segments. The control schema decides the movement to be performed by the motors by analyzing such identified border. The situation is classified according to a predefined set of relevant cases. For instance, if the border is aligned to the reference orientation, if it is biased and the trajectory has to be corrected, if the whole image is composed of wall (non floor) pixels, which means that a turn must be done to avoid collision, etc.

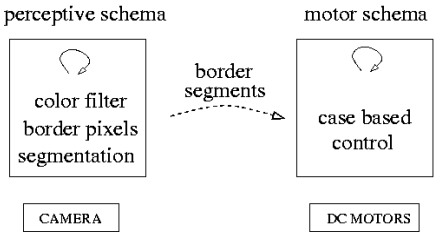

Figure 3: Architecture and structure of the follow-border behavior.

The schemas showed in the figure 3 have been implemented and grouped together in a single program. The program begins with a first phase where all the devices are initialized, and where the distance to the wall to be followed is configured. Once everything has been initialized, three different concurrent threads are started: the first one is the perceptive thread, the second one is the control one, and finally we have added a service thread to monitor the behavior. It checks if the user pressed the END key, and lets the human user to choose among the different visualization options for debugging. This parallel implementation has been feasible because EyeBot operating system offers different facilities for multiprogramming, preemptive and non-preemptive, including concurrent access control with semaphores [**?**][2].

2.1 Perceptive schema

The relevant stimulus identified for the follow-wall behavior is the border line between the floor and the followed wall. The perceptive schema is in charge of finding it and perceiving its main features like position and orientation in the image. This characterization will let the robot know its own orientation to the wall. Processing of images obtained by the robot camera is divided into three steps, which are reflected in the respective functions of the source code. The first one discriminates the field by using a color filter. Next phase searches the filtered image for the points that compose the border line. Third and last phase makes the segmentation of the border line finding the individual segments that compose the line. We will describe these three functions with more detail in next paragraphs.

Color filter. In this first phase the floor is clearly identified, its edges are the border of the field of play. The perceptive invariant found was the floor color. Then, each pixel of the image takes RGB values inside $[0, 255]$, and was classified as *floor* or *non-floor* using the color filter of equation 1.

$$60 >= R <= 140 \qquad 70 >= G <= 150 \qquad 50 >= B <= 100 \qquad R > G \qquad B > R \tag{1}$$

The EyeBot camera offers directly RGB images, so it was a natural decision to use this space to build the color filter. It is important to note, that in order to increase the performance of the algorithm only integer numbers were used. The real numbers processing is emulated in the EyeBot CPU, so it is remarkable slower. Field color characterization was made manually, finding that the values in equation 1 were the appropriate for the regular lightening conditions of our lab. Last two restrictions in equation 1 were added to increase color discrimination.

It is a requirement for the filter to be as robust and efficient as possible. However, RGB space is very sensitive to lightening conditions, as was experimentally proven. One possible solution was to work in HSI space. Probes made in this space shown that the robustness was not increased significantly. In addition the performance was dramatically reduced if the color conversion was done in running time, because EyeBot CPU lacks of mathematical coprocessor to speed up the trigonometric function required for the conversion. The size of a convenient lookup tables required some color degradation, which did not balance the small practical benefits of HSI, so RGB space was finally used.

[2]Manual del robot EyeBot, http://gsyc.escet.urjc.es/robotica/manual-eyebot

Figure 4: Real and filtered image of the environment

Border line. This phase defines clearly the border line, searching for its pixels as the junction between the floor and the wall. The implemented algorithm goes through each column of pixels of the filtered image, looking for the point where the floor ends, which is where the wall starts. It begins from the low part of the image, assuming that the field of play appears at the bottom of the image. In order to make the algorithm more robust to noisy isolated pixels in the bottom part of the image (figure 5), an additional condition was included: in order to consider a pixel as a border one the three next pixels in the column must be non-floor pixels, otherwise such pixels are considered noise and the analysis of the column goes on. This algorithm has been implemented as a finite state machine.

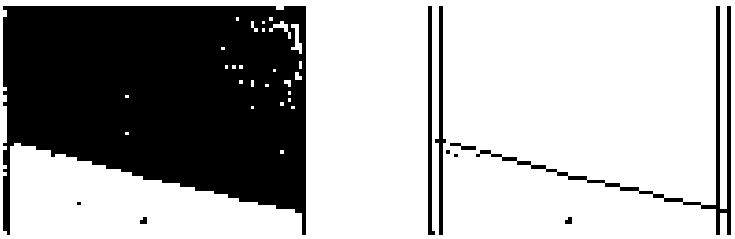

Figure 5: Filtered image and computed border line

Because this pixel classification is not enough to completely characterize the border, it is necessary another processing phase where segments that build up the border line are determined.

Border segmentation. This phase determines the straight segments that make up the border line, their length, slope, etc. If only one segment were detected, the relative orientation of the robot to the wall can be determined using its slope. If two segments with different orientations were found, the robot can infer it is facing a corner.

Different algorithms were tried while looking again for the best tradeof between performance and demanded computing power. Initially several methods based in the first order derivative were tried, because slope discontinuities mean a change of segment. The problem of these methods is that border lines in 60x80 images are not ideal, they are made up by horizontal steps, as shown in figures 5 and 6. This makes very difficult the estimation of the derivative. Additionally these techniques are very sensitive to noisy pixels, which cause large peaks in the derivative. Other technique tried was based on segment growing, where the initial slope is estimated from two initial steps, and then, neighbour steps were merged only if their incorporation does not introduce a big change in the slope. This method was not appropriate because it depends too much on the initial estimation of the slope, which is not always a good estimation of the global one. It caused detection of many segments where the correct segmentation would have produced a single one.

Finally, a new abductive algorithm was developed, based in the verification of hypothesized segments. It extracts segments of the border line from right to left in a very efficient and robust way. As shown in figure 7 it starts hypothesizing a single segment between the first pixel $P1(x_1,y_1)$ and the last one $Pf(x_f,y_f)$ of the border line. Such hypothesis is checked against the intermediate real pixels, from left to right. For all intermediate x their hypothesized ordinate $\widehat{y}$ is computed using equation 2 and compared to the current observed value y. If all of them are closer to the hypothesized line than a tolerance threshold, such segment is validated and it is incorporated to the final output of the segmentation process. Otherwise, when the first pixel out of the tolerance strip appears the hypothesized segment is discarded. Such outsider is a relevant pixel and a new hypothesis is built from that point to the final one. Such new hypothesis is to be verified in the same way.

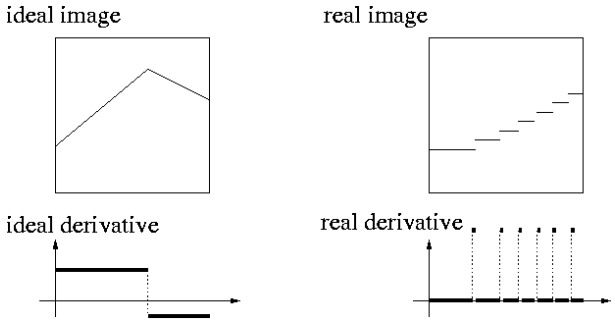

Figure 6: First ideal derivative and the real one of the line

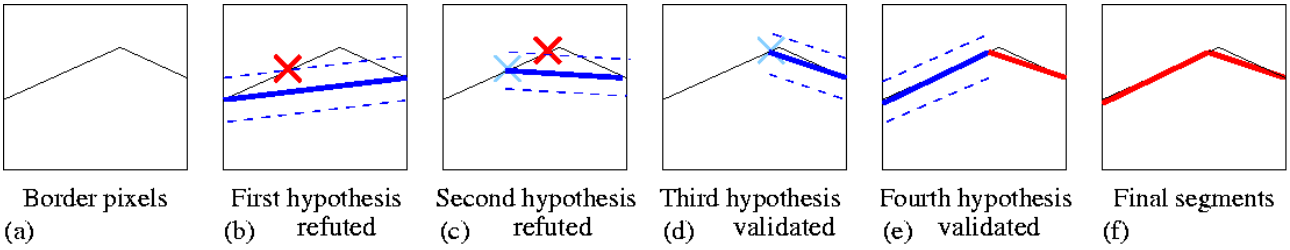

Figure 7: Border segmentation algorithm.

This loop ends with a single validated segment from certain pixel to the one at the right edge of the image, which *explains* such portion of the floor border.

$$m = \frac{y_f - y_1}{x_f - x_1} \qquad \widehat{y} = m(x - x_1) + y_1 \qquad y - y_i < \Delta \tag{2}$$

The algorithm keeps on iterating from the left part of the image until the last unexplained point (beginning of the last validated segment). These operations are repeated until the whole border line has been analyzed, as shown in figure 7. Figure (7a) shows the initial image. Image (7b) shows the tolerance strip and the first unexplained point (marked with a red cross) for the hypothesized segment. The (7c) picture shows another pixel out of the tolerance strip for the second hypothesis. The hypothesized segment from that point to the end is a valid one. Image at (7d) shows the verified segment. The next hypothesized segment, which starts from the left edge and ends in the last explained pixel, is also a valid one, as shown in the (7e) image. The picture in (7f) displays the final two segments found for this border. Despite of the tolerance strip, isolated pixels can make that a single segment looks like two separated ones, divided precisely by the noisy pixel. A final merging step has been included to avoid this problem. In this phase, close segments with similar slopes are integrated into a single one.

This segmentation algorithm obtains reliable and precise information about the robot situation to the wall, as can be seen if figures 8, 9, 10 and 12. Besides, its computational cost is very low. Actually this algorithm has shown the best results in processing time and robustness to noisy pixels in the image. On clear images, which are the most frequent, the segmentation takes only one or two passes to the image, as only one or two segments appear in it. This has been experimentally tested, and the segmentated image rate keeps around 3.5 fps in a regular execution, while the raw image rate ranges to 3.8 fps.

2.2 Control Schema

Control schema is implemented as a separated thread. This thread classifies the border line obtained by the perceptive thread into one of several predefined cases, then it commands the appropiate robot movements for that case. Possible actions of the robot are: go forward at constant speed, turn slowly to the right, turn slowly to the left, turn firmly to the right, and turn firmly to the left. The following cases have been identified, taking into account that the robot want to follow the border on its right (symmetrical cases can be used for a border on its left):

Figure 8: Filtered image and segmented border line

- *Infinite wall*: The robot sees the wall as in figure 9, that is, it finds only one segment, whose orientation can be analyzed. This orientation is obtained by analyzing the end points of the segment, which is more reliable than the single slope. If the right end is higher than the reference one, that means that the robot is losing the wall, the control decision then is to proportionally turn towards the wall. In the opposite case, when the right end is lower than the reference, then the robot is getting closer to the wall, and it should turn away proportionally to the difference. Finally, if the point is close to the reference pixel then the robot is aligned and it simply has to keep on forward.

Figure 9: Segments in the infinite wall case.

- *All-floor or all-wall*: Both cases are instances of the extreme situation where the border line is just a single horizontal segment in the top (everything is floor) or the bottom of the image (everything is wall). In the first case the robot must turn firmly right looking for the wall to follow, in the latter it should turn left tightly to avoid the collision.
- *Concave corner*: When the robot is approaching a concave corner the processed image is similar to that shown in figure 10. The robot perceives at least two segments, the one on its right, the key segment, and another one that departs from its left end. We call it the key segment, because it is the most relevant one to the robot, considering it has to follow the wall on its right. If such key segment is long enough then the robot can go on straight ahead, because there is still enough wall to be followed. If its length is smaller that means robot is near the end of the lateral wall, and the frontal one is being reached. In such a case the robot has to turn left to avoid the collision.

Figure 10: Segments in the concave corner case.

- *Discontinuity*: In this case the border line is made by a great number of segments, and most border pixels don't have any segment that explains them. One example of this situation is the figure 11. This situation appears when the images captured by the robot are too noisy, or when the lateral wall is too far. So, the control action should be to recover the wall turning right.

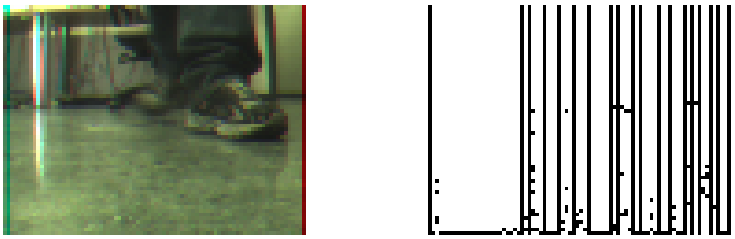

Figure 11: Segments in the discontinuity case.

- *Convex corner*: This happens when besides the key segment there is a vertical segment, as illustrated in figure 12. Such vertical line reflects the followed wall finishes there. In this case the robot should turn right when the lateral wall end is close enough, that is, when the key segment were small enough. This case never appears in the robocup follow-border scenario, and so has not been implemented, but has been included for the sake of completion.

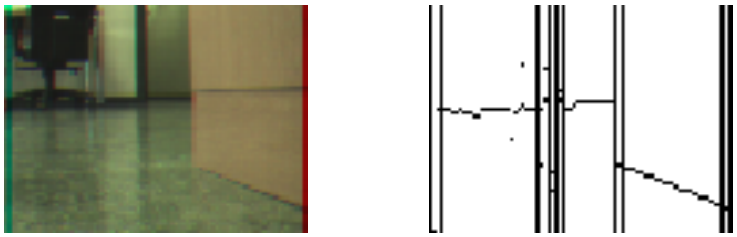

Figure 12: Segments in the convex corner case.

3 Follow-ball behavior

Two additional schemas are developed to generate the follow-ball behavior. In order to ease its development and debugging they were combined together in a single program as shown in figure 13. Obviously, the relevant stimulus in this case is the ball and its relative position to the robot. First schema is a perceptive one, in charge of characterizing the ball in the image, its position and size. Second schema is a control one, that uses that information to command the robot movement. As well as in the follow border behavior, this design has been implemented using three parallel threads: one carries out the perceptive algorithms, other the control algorithms, and the third one to manage the user interface. The last service thread checks the exit button, and let the human user to choose among several visualization options for debugging.

Liveliness is the essential requirement in this behavior. In this way it is necessary to perceive images as fast as possible to ease a good ball following. If the ball is moving the robot need to realize it as soon as possible, in order to correct its own motion, keep the ball close and do not loose it. The designed tracking control is a reactive controller with a very short control loop. In this case, the potential (and real) bottleneck is the perceptive thread, because it involves the costly image processing. Its processing speed determines the maximum ball velocity that the robot is able to follow properly.

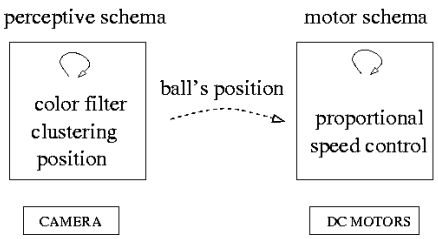

Figure 13: Schemas for the follow ball behavior

3.1 Ball perception

The ball chosen for the experiments was an orange golf ball, which complies the Robocup rules and is easy to detect. The perceptive schema has been implemented as a thread executing an infinite loop. It captures the current image and searches into it for the orange ball, returning its position and its pixel size. The first step is to discriminate the loud orange pixels using a color filter. In a second step close orange close pixels are clustered together. The mean value of the cluster is only an initial guess of ball position, because the orange stain may include only part of ball pixels, depending on the lightening. If needed, a third step runs an ad hoc algorithm developed to correctly estimate real ball position in the image.

The perceptive thread works in an efficient infinite loop that let it reflect into the perceptive variables the relative position changes of the ball. Maximum capture rate in the EyeBot robot is 4 frames per second, which was reduced to 2-3 fps. when the perceptive processing was included. This speed only let us follow balls which move slowly, and that is a major limitation in our current football team. In the next subsections the details of the perceptive steps are described.

Color filter. This operation is made using a RGB filter that picks up orange pixels only. The reasons to use the RGB space instead of other color spaces are the same that in the previous behavior. In the same way that in the follow border behavior, values of the ball filter are reflected in the equation 3.

$$99 >= R <= 255 \qquad 65 >= G <= 206 \qquad 48 >= B <= 184 \qquad 1.25 > R/G < 1.8 \tag{3}$$

Additionally, another threshold has been added: if the number of detected orange pixels is less than this threshold, then it is concluded that there is no ball in the image. Such case appears when the ball has been lost and so it is out of EyeBot's field of view. Then, no more calculi are made in the perceptive schema.

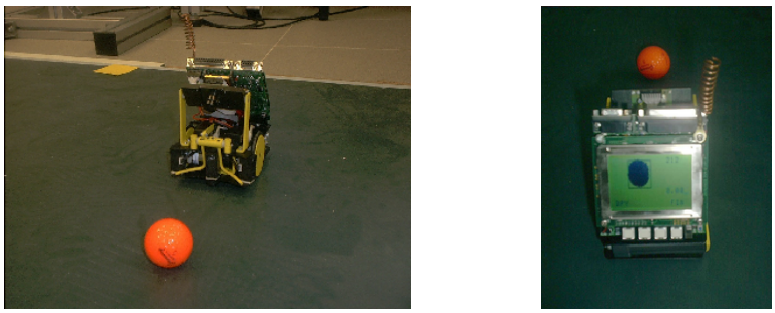

Figure 14: Ball perception

Clustering. The color filter only classifies the pixels as orange or not-orange, but it does not say where the ball is, neither how many balls are in the image. In order to do so, the orange pixels have to be grouped into different disjoint clusters. The groups are identified looking for areas with high density of orange pixels, analyzing the histogram in X and Y axes.

First, the histogram of the X-axes is computed, containing the number of orange pixels in each column. An histogramic threshold is used to determine the starting column and the end of the each zone. When the density of pixels cross over the threshold the beginning of a zone is marked, and when it goes below the end is also noted down, as shown in right side of figure 15. In the same way, the edges of the cluster in the Y-axes are computed looking at the histogram of Y-axes, which contains the number of orange pixels in each row. Actually the algorithm analyzes the vertical and the horizontal axes independently, so the groups are modelled as rectangular windows.

Such single histogramic threshold has to be high enough to discard noisy pixels, so spurious points do not generate enough density to surpass it. It also has to be low enough to catch balls of certain size in the image. This single histogramic threshold technique has the disadvantage that reduces the ball zone, that is, the edge pixels are considered out of the ball, as shown the left part of figure 15. This problem can be solved using a double histogramic threshold technique. The upper threshold ensures that there is a relevant number of pixels, meaning that an orange cluster has already started. The lower threshold expands that border to

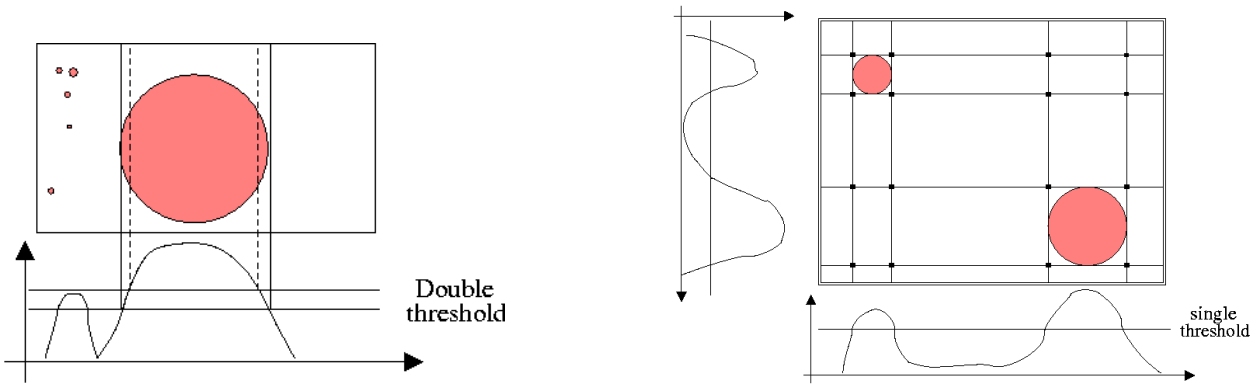

Figure 15: Double threshold (left) and grouping of two balls using single threshold (right)

the real beginning of the zone. The end of the cluster is also expanded in a similar way. In the current implementation, lower threshold has been set to 2 pixels, and upper one to 4.

After this step we have different horizontal segments with the first and last columns of some cluster, and different vertical segments with the first and last rows of such clusters. To get the real clusters every vertical segment has to be combined with every horizontal one, forming a rectangular window, which hopefully contains the ball pixels. Each window corresponds to a possible ball and is characterized by its four corners. In the case of getting only one ball, two segments are obtained, one in X axes, and other in Y axes. If several segments are obtained, then more than one possible ball are identified and several ghost windows may appear in the image, as shown in the right part of figure 15.

This hallucination is due to treating independently both axes. Despite this drawback it is still convenient to compute both axes separately, because the performance is greatly increased. To discard ghost clusters an additional threshold is also introduced, windows having less pixels than that threshold are filter out. So the final outcome of the algorithm is a collection of rectangular clusters, which do contain orange balls.

In the case of several true balls appearing in the image, the cluster containing the greater number of orange pixels is chosen as the target to follow. Its position and size in the image are the output of this perceptive schema, and the information provided to the control schema of the behavior.

Ball position and size. The position and size of the stains are related with the ones of the balls, but they are not an exactly reflection. This happens because not all the areas of the ball do receive the same amount of light, and so, some of its pixels do not pass the color filter, as shown in figure 16.

In order to calculate the right size and position of the ball in the image we have developed and ad hoc algorithm. The spherical nature of the ball is very helpful here, it can be assumed that the ball always will appear as a circle in the image, regardless its pixels pass the color filter or not. In addition the extreme pixels of the cluster, those which have the maximum or minimum x, or the maximum or minimum y, belong to the outer circumference of such circle. From these four points the center of the circumference can be estimated, as well as its size. The equation of the circumference can be solved using only three (P_1, P_2, P_3) of these four points. Following equations obtains the center (x_c, y_c) and the radio R of the projected circumference:

$$x_c = \frac{-y_2y_3^2 - y_2x_3^2 + y_1y_3^2 + y_1x_3^2 - y_1x_2^2 - y_1y_2^2 + y_2y_1^2 + y_2x_1^2 + y_3y_2^2 + y_3x_2^2 - y_3y_1^2 - y_3x_1^2}{2(x_1y_2 - x_1y_3 + x_2y_3 - x_3y_2 - y_1x_2 + y_1x_3)} \tag{4}$$

$$y_c = \frac{-x_2y_1^2 - x_2x_1^2 - x_3y_2^2 - x_3x_2^2 + x_3y_1^2 + x_3x_1^2 - x_1y_3^2 - x_1x_3^2 + x_1x_2^2 + x_1y_2^2 + x_2y_3^2 + x_2x_3^2}{2(x_1y_2 - x_1y_3 + x_2y_3 - x_2y_1 - x_3y_2 + y_1x_3)} \tag{5}$$

$$R = \sqrt{(x_1 - x_c)^2 + (y_1 - y_c)^2} \tag{6}$$

Despite being more accurate, this algorithm demands some computations which slow down the perceptive schema speed. This prevents liveliness in the tracking behavior, because the robot do not have fast feedback about the effect of its movements in relative ball position. Due to the low computing power available in the EyeBot platform, it was decided to compute the center of the ball as the geometric center of the rectangular cluster, whose calculus is simpler and faster.

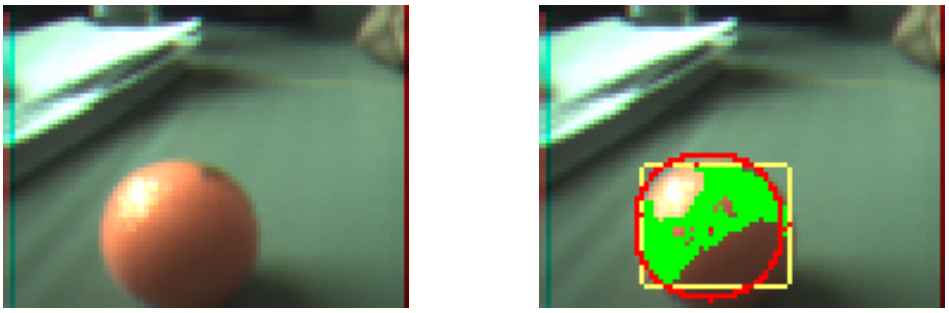

Figure 16: Orange points and ball position and size estimation

3.2 Control schema

The control schema was implemented as a thread that closes the feedback loop. The position of the ball in the X axes feedbacks the turning of the robot, and its position in the Y axes feedbacks the forward motion. If the ball appears in the central pixels of the image, then it is facing the robot, so no movement is needed. If it appears in other areas the motor control tries to correct the ball deviations, moving the robot in order to center the ball in the image. For instance if the ball moves upwards in the image that means that the ball is going away, so the robot has to speed up to pursue it. Obviously, if the ball moves downwards the robot has to slow down, or even going backwards. Simultaneously, if the ball move to the right side of the image, then the robot has to turn in the same way to track it.

Two differents controllers have been developed and compared to carry out this function: a speed control and a position control. In both cases the friction and the possibility of oscillations or unstabilities need to be taken into account.

Position control. The position control uses the sign of the difference between the position of the ball and the center of the image. If the ball's y is over the center, a small forward distance is commanded. If it is below, a small retreat is ordered. The same strategy is used analyzing the angle: if the ball's x is on the left, a small turn is made in that orientation; If ball x is on the right, a right turn is commanded.

To perform the real movements of the robot we use the low level routines provided by the manufacturer, which use fast PID control loops over PWM signals. These routines are blocking, that is, they do not return control until commanded movement has been finished. This means, for instance, that turns and advances cannot be made simultaneously. So, the control thread executes in each iteration a small turn (2 degrees) and a small advance (0.5 cm.) according to the last image analyzed. The amplitude of such steps have been experimentally tuned, small enough to make the robot not oscillate around the ball. The global trajectory appears as a sequence of small steps: turn, advance, turn, advance,etc. This makes this controller be very slow, but very stable. Another consequence in this particular case is that it is necessary to synchronize the perceptive thread and the control one in such a way that only one action is carried out for each image taken.

Speed control. Position control is not able to track moving balls, so a speed control was developed to get a follow-ball behavior when the ball was moving. This control has been implemented using non-blocking low level routines, which are able to make fast corrections on the motors PIDs. These routines provide simultaneous turn and advance movements, and also let the user to change the reference speeds at any time.

A proportional controller has been developed to correct deviations of the ball from the center of the image. Again, the horizontal deviations only affect the turning speed, and vertical ones to traction speed. The control profile used to correct traction speed is shown in figure 17. A similar one is used for rotation velocity. There is a central zone where the robot is considered correctly aligned, and so the commanded speed $V_{commanded}$ is null. If the deviation goes over the threshold L_{min} then a proportional speed is commanded to correct it. This L_{min} threshold avoids the oscillations around the ball position preventing any correction when the ball is close to the center. Another threshold L_{max} is used also to avoid overcorrection.

The maximum V_{max} and minimum feedback speeds V_{min} greatly depend on the surface where the robot is moving. If the surface is rough, the speeds have to be higher than over smooth surfaces. Additionally, charge batteries has to be taken into account, if they are completely charged speeds should be smaller. All these parameter values have been experimentally tuned for the official RoboCup field to $L_{min} = 30 \quad pixels$ for turnings, $L_{min} = 20 \quad pixels$ for traction. V_{min} has been set to 30% of the V_{max}.

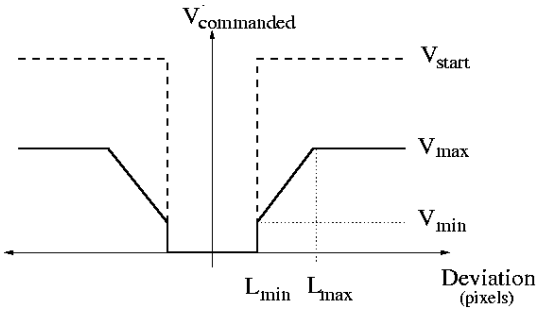

Figure 17: Feedback profile in the speed control

The control profile shown in figure 17 was used to correct robot speeds when the it was moving. Nevertheless such profile did not work when the robot was stopped. If control values were tuned for dynamic motion, then those feedback speeds were not high enough to start the motion. Besides, if control values were tuned high enough to start the motion, then such profile caused high oscillations when the robot was moving, and the ball was easily lost. The underlying problem here is the different values for dynamic and static friction, latter being much higher than the first one. To solve such trouble we introduced another feedback profile when the robot was detected to be stopped. It is shown in figure 17 with dashed lines. It has a central zone without any motion required and out of that, a speed overshooting V_{start}. The idea is to command a speed peak able to win the static friction and really start the robot motion. After that the robot is moving and the regular control profile is applied. Detection of the robot being stopped is done using encoder readings, comparing the last value with the previous one. If the difference is less than a threshold then the robot is stopped and V_{start} is used. Using this doublesided profile care is to be taken that V_{min} is able to surpass the dynamic friction. Otherwise the robot will go jerkly forward, jumping from static overshooting to V_{min}, which makes it stop, and then overshooting again.

Comparing the performance of both controllers, speed and position controls, the results show that speed control is slightly more unstable when the ball is quiet, but it let the robot follow moving balls.

4 Conclusions and further work

The first goal of this work was to probe whether DSH architecture was suitable for robots with small processing capabilities. Two behaviors, follow-border and follow-ball, have been successfully implemented [3] complying the schema distribution of DSH. They show that this architecture can be used in low-class robots. In particular, perception and control partition into different schemas has been validated, as both behaviors have been implemented using two schemas: a perceptive one, which extracted the relevant stimulus for the behavior, and a control one, which made use of such information to choose the right movement commands.

The implementations developed are lively and make their job analyzing more than 3 frames per second, which is very efficient, since the hardware rate is 3.7 fps. They have been greatly influenced by the hardware limitations of the EyeBot robot. In particular, the lack of mathematical coprocessor has forced the use of simplified algorithms (for instance the use of only integer numbers), taking always into account that perception has to be as fast as possible. In the same way, all color filters work in the RGB space, others spaces have been considered (HSI, LAB, etc.), but hardware constrains have prevented their use.

Concerning the behaviors themselves, the first one, follow-border, was implemented using a novel very efficient abductive algorithm to determine the border line between the floor and the wall. The control schema was based in cases, distinguishing continuous walls, concave corners, convex corners, etc. The second one, follow-ball, was implemented in a similar way, using in this case an histogramic clustering algorithm, complemented by several thresholds to improve its efficiency. In its control part a reactive controller was implemented, taking care of the special characteristics of the EyeBot in particular the static friction. It is important to notice that the correct selection of the parameters is a crucial point in order to obtain the desired behaviors.

Both behaviors use local vision and are very useful in a robocup scenario. Actually they are the stepping stone of our small league soccer team. Further works in this area will also focus on adding new behaviors to the soccer players and improving existing implementations. For instance follow-wall behavior should be able to cope with obstacles, and using the infrared sensors would probably improve its performance. The use of some

[3] Source code of the programs and some video demos are available at group's web page `http://gsyc.escet.urjc.es/robotica`

prediction in the movement of the ball, would also improve the performance of the follow-ball behavior. We are also working in position estimation inside the game field from local images using Montecarlo techniques.

In addition, the two developed behaviors do not completely probe the full features of DSH, mainly because no dynamic hierarchies have been required for them. We intend to develop new more complex behaviors in the robocup scenario to further probe more features from DSH, like the schema combination and compound stimuli perception.