Real time game field limits recognition for robot self-localization using collinearity in Middle-Size RoboCup Soccer

Fernando Ribeiro (1)
Gil Lopes (2)

(1) Department of Industrial Electronics, Guimarães, University of Minho
(2) Department of Computer Sciences, Porto, Portucalense University

Keywords: Mobile Robotics, RoboCup; self-localization; collinearity; edge detection.

Abstract

Enabling a mobile robot to achieve its self-localization in real-time with vision only, demands for new approaches and new computer algorithms. An approach for giving game field self-localization to a Middle Size RoboCup Soccer robot can be based in two steps: finding the game field lines and evaluating the obtained coordinates calculating the robot coordinates. This paper describes a method to achieve the first step. This approach is based on an algorithm that combines three major features: edge detection, selection and collinearity search. The final target is to retrieve the line segments (defined by its two limits coordinates), which identify the game field boundary lines. These line coordinates will be used on the next step that is the process to calculate the robot position in the game field. Since this first step is to find lines in real time, it is an alternative method to the Hough Transform Method.

1. Introduction

Computer vision in Middle-Size RoboCup Soccer is greatly responsible for success in this competition. Apart from other technologies that some teams use to achieve robot location/orientation (a description is made by [5], most of them are based on computer vision and recognition. But robot orientation does not mean robot localization. MINHO team [6] uses cameras for giving robot orientation even though the robots do not know where they are on the field. They simply follow a purpose of making a red ball to be aligned with a goal (blue or yellow) and kick the ball, avoiding collisions with any obstacles (black or white entities on the field). Since the robots do not know their location on the field, it is difficult to play in cooperation with the rest of the team. Moving around the game field is based only on virtual colour sensors detection that permits also avoiding obstacles such as the other robots or the game field walls. Recognizing the game field limits may be the first step for self-localization acquisition.

Using cameras and grabbing images for post-processing, requires most of the CPU processing time for this task. The need for using fast algorithms with minimum processing time makes software developers to create/develop new ideas for implementing known methods or creating new ones. One method that achieves the proposed objective, is based on the Generalization of Hough Transform by Ballard [1]. This is the most used in line detection, but in a real time environment this implementation could compromise all system performance [2].

Indeed, there are lots of tasks that the processor needs to do like monitor the IR sensors, instruct the robot actions, communicate with the other robots, grabbing images, process strategy program, watching the batteries charge, and so on. In each machine cycle, most of these tasks must be fulfilled. The remaining processing time (very little) is for image processing. Depending on the type of robot construction and computer platform, this time may not be long enough to perform all the tasks. That is the reason why the image processing algorithms must be carefully designed and implemented.

For this work the image size used for each frame is 320x200 (256 grey scale) pixels. The image size choice is of extreme importance because too much resolution brings down the algorithm performance and too low resolution may raise losses of information. The image size used for the captured images proved good results output.

The main reason to use grey scale rather than colours was that, to find segments or lines in an image, it is only a matter of image transitions in contrast. There is no advantage in using colours since the grey scale can give those transitions too with less computer effort. In this way, it is better to avoid the use of colours for this purpose. Even though, grey scale could give sharp transitions, making the edge detection more complex.

The picture below is the original image used all over this paper to describe the work, and consists of a scene of a robotic football game.

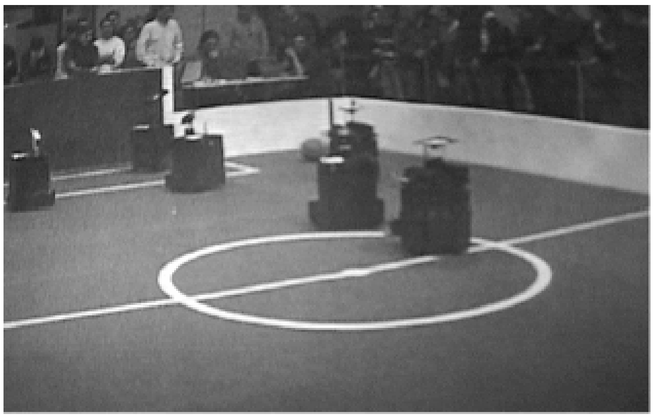
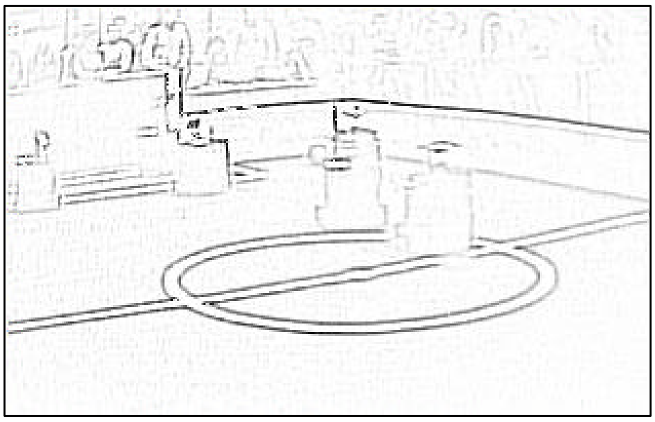


Fig. 1 - The original 320x200 grey scale picture.

2. Edge and point detection

Finding edges in the picture is the first step for using this method. Convolution masks [3] are the best way to explain how this was achieved. These masks give the image contour; unfortunately the calculations have to be carried out on the whole image.



$$\begin{vmatrix} 1 & 0 & 0 & 1 \\ 0 & -1 & -1 & 0 \\ 0 & -1 & -1 & 0 \\ 1 & 0 & 0 & 1 \end{vmatrix}$$

Fig. 2 – Resulting image after applying the 4x4 convolution mask on the right.

This task is very time consuming due to the pixel-by-pixel mask calculation. The algorithm performance can be greatly improved by calculating only the points needed. The idea is to draw virtual lines (referred to as **detection lines** onwards), beginning at a central point on the [*x*] axis and with an extreme [*y*] axis, as described in fig. 3.

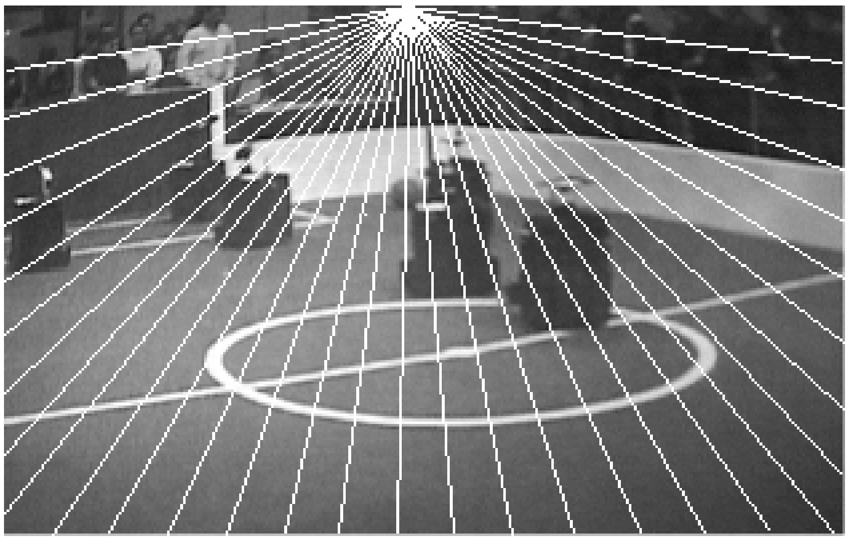


Fig. 3 – Original picture with the detection lines.

When each line is drawn, the grey value of the current pixel is subtracted from the grey value of the previous pixel. If the result is greater than a trigger value, then it is considered a transition on the image, and so, it means that a relative point was found.

The trigger value must be user adjustable because it depends on the environment light intensity at the game field. For this example, a value of 50 was used with good results, as shown on the next figure.

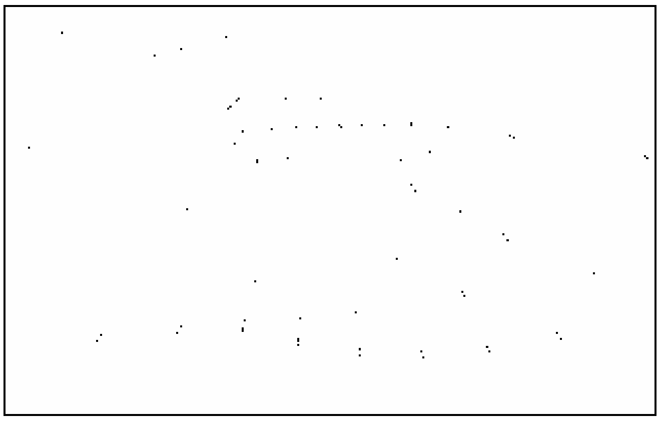


Fig. 4 – Resulting points after the detection lines process

3. Point selection

The resulting image after applying the **detection lines** has some redundant points. Several points detected from the same feature are not to be selected. These points may be originated by noise on the image or by sudden variances in the features captured. This amount of points could compromise the reliability and performance on the next step. Therefore, a point selection must be carried out. To filter these points, a **selection grid** is used which acts like a filter for these disposable points. The grid size must be user adjustable and it means that in the 320x200 image size, with a 20 pixels size cell, the result would be a 16x10 grid. The grid cells are composed by Boolean values and cleared each time a new image is processed. The tests showed good results with a 20 pixels size cells for this application.

The selection grid is filled during the process of point detection with the detection lines. When a new point is found, the [x, y] coordinates are divided by the size of a cell. The values returned match with one of the cells in the grid. For example, if a point coordinates are [120,80] with a 20 pixels size cells, the matching cell will be the cell [6,8]. If the Boolean cell value is false, then this value is toggled and the point is stored in a vector with the original [x, y] coordinates and the angle from its detection line. If another point matches the same cell, it will find a Boolean cell value of true and it will be ignored. At the end, the **resulting vector** will have just one point from each matched cell. The next figure shows the resulting points after applying this filter.

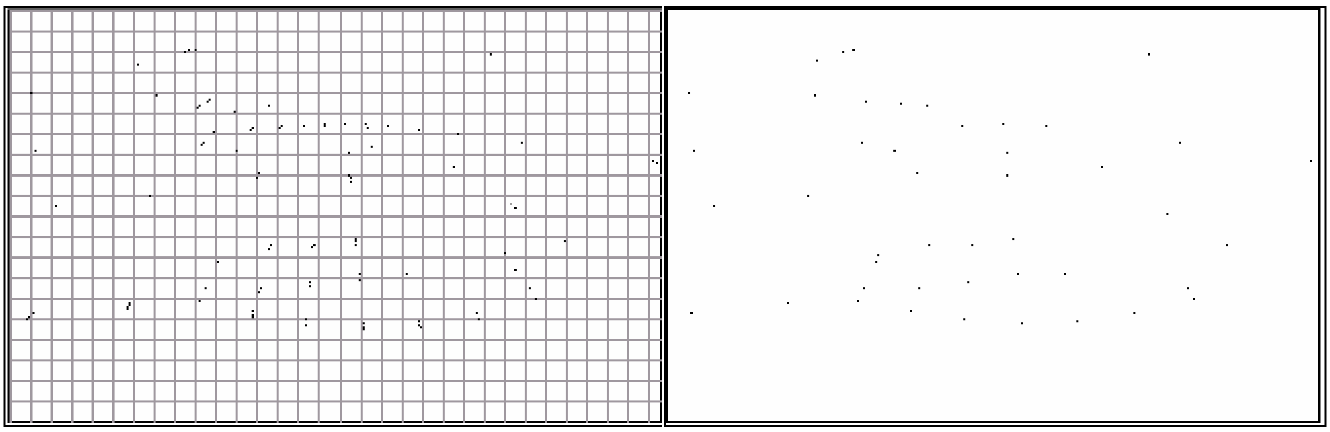
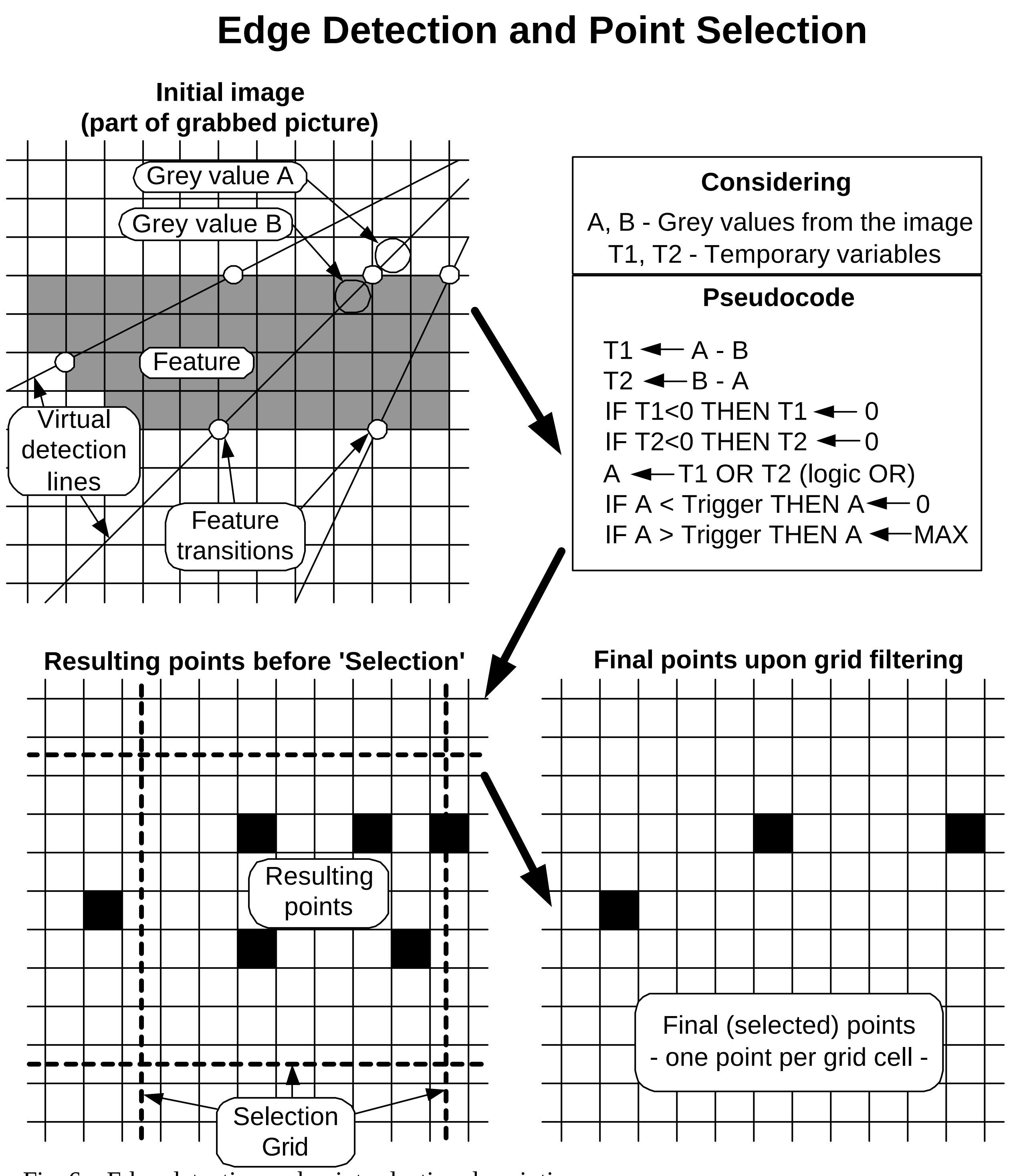


Fig. 5 – Selection grid on the left and the resulting filtered points on the right.

It is well perceptible a lower density of points that result from this detection, without great loss of information. After finishing the points gathering, the resulting vector needs to be sorted. The sort must be made first on the [*y*] coordinate and then on the [*x*] coordinate.

The following figure describes the whole process.


Edge Detection and Point Selection
Initial image (part of grabbed picture)
Grey value A
Grey value B
Feature
Virtual detection lines
Feature transitions
Considering
A, B - Grey values from the image
T1, T2 - Temporary variables
Pseudocode
Resulting points before 'Selection'
Resulting points
Selection Grid
Final points upon grid filtering
Final (selected) points
- one point per grid cell -

```
T1 ← A - B
T2 ← B - A
IF T1<0 THEN T1 ← 0
IF T2<0 THEN T2 ← 0
A ← T1 OR T2 (logic OR)
IF A < Trigger THEN A ← 0
IF A > Trigger THEN A ← MAX
```

Fig. 6 – Edge detection and point selection description.

4. Finding collinear points

To find the collinear points from the resulting vector, the algorithm makes a search for all the combination of each three points, with different angles. Points with the same angle very likely belong to the same virtual detection line, and therefore they are naturally collinear. Even though this could really happen, it is preferable to ignore the resulting segment, since the robot is moving constantly and in the next frame, it will most certainly detect the correct line.

For each set of three valid points found, a test for collinearity is made. This test is based on the calculus of the following 3x3 determinant:

$$\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix}$$

where (x_1, y_1), (x_2, y_2) and (x_3, y_3) are the coordinates obtained from the resulting vector of the three points for the collinearity test. Using the *Sarrus* rule [4], the value is given by:

$$d = (x_1 * y_2) + (x_2 * y_3) + (x_3 * y_1) - (x_2 * y_1) - (x_3 * y_2) - (x_1 * y_3)$$

If the matrix determinant (*d*) is 0 (zero), the three points are collinear. This is the ideal case, but not always it happens and a **tolerance** is accepted. If one of the three points is slightly deviated from the line where the points are settled, the tolerance considers its deviation. The tolerance used in these tests (and proved good results) was 5.

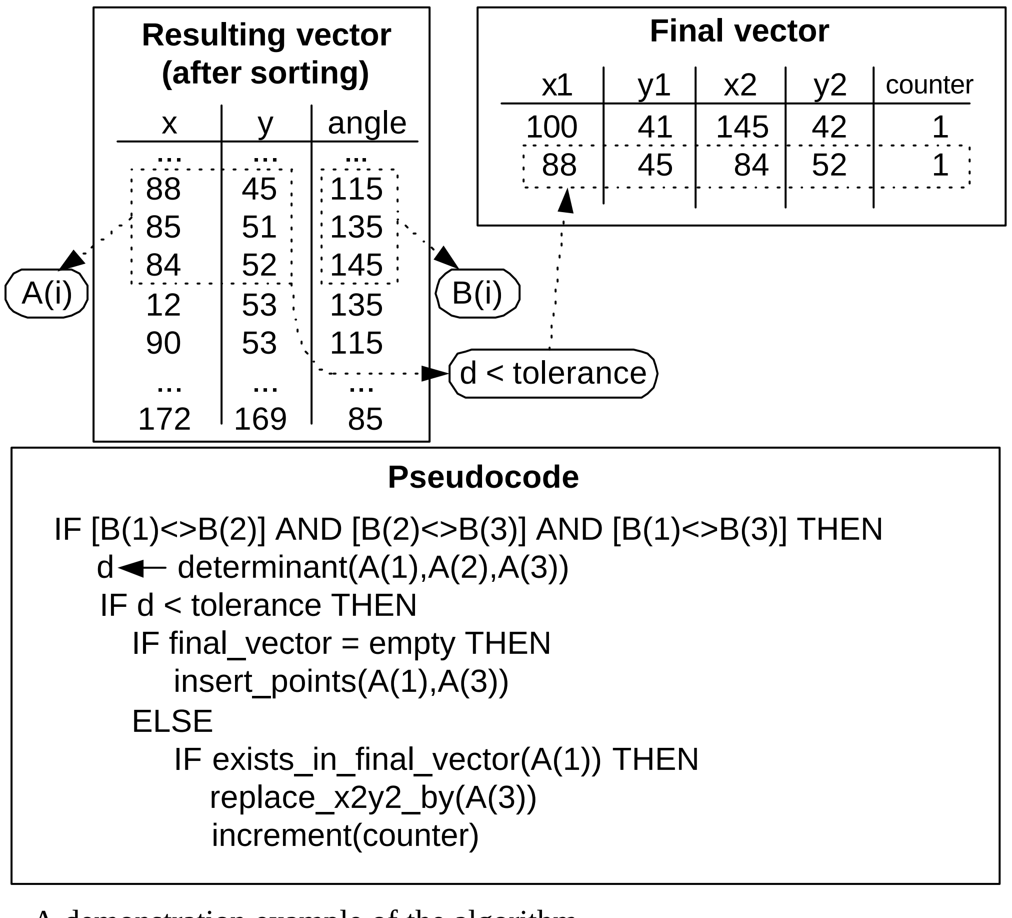
After a positive test of collinearity, the two extremes of the three coordinates are stored in the **final vector,** because since the points in the resulting vector are sorted, the first and the third point are the most distant ones. They represent one segment that was found.

The final vector has three fields for storing the segments:

1- Start point
2- End point
3- Counter

The counter field gives the number of collinearity values found for that segment. In other words, when a segment is ready to be stored, it has to be checked whether it is not a part of another segment already stored. This is carried out with the first point of the new segment. If it is already in the vector, it means that the third point is a more distant point than the second stored in the final vector. In this case it is only replaced the second point and incremented the counter. This means that the segment is bigger than the one stored before. This is only possible because in this stage the resulting vector is sorted and this searching process for collinear point starts from the beginning of the vector. If the first point does not belong to the final vector, the first and the third points are inserted in the vector and the counter is reset to 1 (one) if they are not collinear with the other points already there. This is a test that needs to be carried out point-by-point.

The following picture describes the steps to implement this procedure.



Resulting vector (after sorting)

x	y	angle
...	...	...
88	45	115
85	51	135
84	52	145
12	53	135
90	53	115
...	...	...
172	169	85

Final vector

x1	y1	x2	y2	counter
100	41	145	42	1
88	45	84	52	1

Pseudocode

```
IF [B(1)<>B(2)] AND [B(2)<>B(3)] AND [B(1)<>B(3)] THEN
    d ← determinant(A(1),A(2),A(3))
    IF d < tolerance THEN
        IF final_vector = empty THEN
            insert_points(A(1),A(3))
        ELSE
            IF exists_in_final_vector(A(1)) THEN
                replace_x2y2_by(A(3))
                increment(counter)
```

Fig. 7 – A demonstration example of the algorithm.

After seeking the collinearity points, the resulting vector could be demonstrated in the following figure.

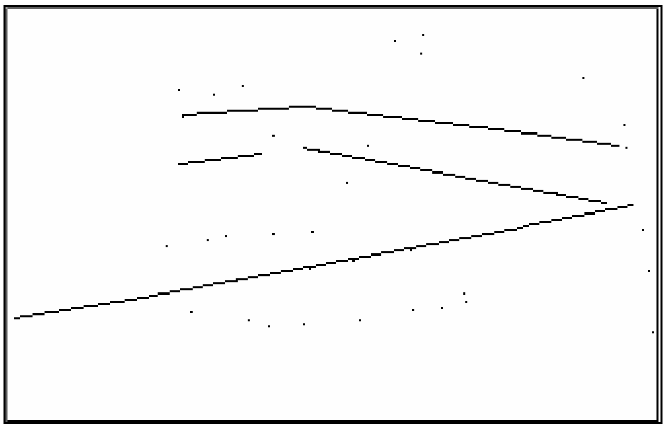


Fig. 8 – Detected segments after processing the algorithm

In this case, the counter could give the major segments, since this value means the number of small segments that comprises the final segment. The greater the value, the greater the segment. This means that the field limits or the game field lines are the segments with the greatest counter. The next step would be the translation of these segment coordinates to robot localization. This will be discussed in future papers.

Conclusion

This method proved to be promising with good results, and demands less effort in calculation and memory consumption from the computer. All the calculi were optimised to the maximum to minimize or even avoid the use of trigonometric functions and other kind of time consuming mathematical functions. These functions take from the processor too many machine clocks delaying the whole process. Using an Intel® x86 platform with a Pentium II at 350MHz, a Brooktree® BT848 PCI frame grabber board and MS-DOS® operating system, with the development environment in C and Assembly, the final achieved performance was 12 images/sec processed. This could be significantly improved with better processor clocks, but could also mean higher power consumption. There are always several commitments that the developer must be careful for and this method could help in some decision taking.

Processing captured frames in a real time basis is a task that has to be accomplished as quickly as possible because time is critical. Speeding the process or having more time for the process described, means increasing the quality of the output generated. Since this paper only describes the first step for achieving robot localization, this output must be passed to the next step with substantial information and although there must be enough spare time to process the next localization tasks.

References

[1] D. H. Ballard, "Generalizing the Hough Transform to Detect Arbitrary Shapes," Pattern Recognition, Vol. 13, No. 2, pp. 111-122, 1981.
[2] Markus Ulrich, C. S., Albert Baumgartner, Heirich Ebner (1998).- “Real-Time Object Recognition in Digital Images for Industrial Applications.” - http://wwwradig.informatik.tu-muenchen.de/papers/2001/Optical-3D-2001-Ulrich-etal.abstract.html
[3] Young, D. (1993). - “CONVOLUTION.”: - http://www.cogs.susx.ac.uk/users/davidy/teachvision/vision2.html
[4] Chambers, Trevor (2001) – “MATHEMATICS FOR COMPUTER SCIENTISTS” – University of Warwick - http://www.dcs.warwick.ac.uk/people/academic/Trevor.Chambers/sarrus.pdf
[5] Sérgio Monteiro, Fernando Ribeiro, Paulo Garrido, "Problems, Solutions and Trends in Middle-Size Robot Soccer - A review", Robotica'2001 - Festival Nacional de Robótica, 25-28 Abril 2001, Guimarães, Portugal.
[6] Carlos Machado, Sérgio Sampaio, Bruno Martins e António Ribeiro, “Image Processing Applied to a robotic Football Team”, Workshop on EuRoboCup’2000, Amsterdão, Holanda, 28 Maio – 2 Junho, proceedings em CD-Rom, Springer.