Soccer Field Recognition Using Normalized Image Technique

Débora T. Valverde, Gabriel de B. Silva, Gustavo R. Silva, Heitor M. S. Cunha, Iago A. Peixoto, João V. G. Silva, Leonardo L. R. Barbosa, Renata Bernardes and Rogério M. R. Filho

***Abstract*—This paper represents the application of a program to identify the soccer field. This program was made to normalize an image and with it we can have a more trustworthy image and identify the field with safety. This program also minimizes the errors generated by shadows, brightness and contrast, allowing the user to obtain the same result despite these factors.**

I. Introduction

This work was developed by EDROM(Equipe de Desenvolvimento de Robótica Móvel), a mobile robotic development team, that does researches about humanoids robots and LEGO robots in order to increase the students' knowledge in various fields of mechatronics engineering, furthermore, the project is used to compete at events around the Latin America and also sometimes out of Latin American, at world competitions like the RoboCup[1].

The program, made in C++ and using the open source library: OpenCV[2], for the robotics competition RoboCup Humanoid Soccer, that requires a fully autonomous robot that can identify the field zone and play a soccer match. It allows the robot to identify the field with more precision and reliability. It was made to identify the field out of an image. There is a function in this program that minimizes the errors found when there are certain changes in the image, like a change of brightness, shadows, and other factors that could change the tone of the field.

Débora T. Valverde is a Mechatronics Engineering student of the Department of Mechanical Engineering (FEMEC) from the Federal University of Uberlandia (UFU) (e-mail: debora_tahara@hotmail.com).

Gabriel de B. Silva is a Mechatronics Engineering student of the Department of Mechanical Engineering (FEMEC) from the Federal University of Uberlandia (UFU) (e-mail: gabriel_gx7@yahoo.com).

Gustavo R. Silva is a Mechatronics Engineering student of the Department of Mechanical Engineering (FEMEC) from the Federal University of Uberlandia (UFU) (e-mail: gustavorezendesilva@hotmail.com).

Heitor M. S. Cunha is a Mechatronics Engineering student of the Department of Mechanical Engineering (FEMEC) from the Federal University of Uberlandia (UFU) (e-mail: heitormenezes10@gmail.com).

Iago A. Peixoto is a Mechatronics Engineering student of the Department of Mechanical Engineering (FEMEC) from the Federal University of Uberlandia (UFU) (e-mail: iagoaph@gmail.com).

João V. G. Silva is a Mechatronics Engineering student of the Department of Mechanical Engineering (FEMEC) from the Federal University of Uberlandia (UFU) (e-mail: joaovitor_g.s@hotmail.com).

Leonardo L. R. Barbosa is a Mechatronics Engineering student of the Department of Mechanical Engineering (FEMEC) from the Federal University of Uberlandia (UFU) (e-mail: leonardobarbosa@meca.ufu.br).

Renata Bernardes is a Mechatronics Engineering student of the Department of Mechanical Engineering (FEMEC) from the Federal University of Uberlandia (UFU) (e-mail: renata.bernardes@ufu.br).

Rogério M. R. Filho is a Mechatronics Engineering student of the Department of Mechanical Engineering (FEMEC) from the Federal University of Uberlandia (UFU) (e-mail: rfilho_11@hotmail.com).

The function analyzes the image and highlights the RGB colors (Red, Green and Blue) by normalizing pixel by pixel. By doing so the highlights effect occur and the user can distinguish more easily the color green, which is the color of the field, and can be used to identify the other two colors with precision.

After processing the image with this function the program identifies the areas with the green color and uses the bigger one, so the program assumes that the bigger green area is the field.

II. Normalize the Image

In order to understand the program, it is needed to understand what the function mentioned above do.

The pixel, the smallest unit of an image and it can be represented by values in the three colors: Red, Green and Blue. So each pixel has 3 values in each color, these values goes from 0 to 255 and they represent the intensity of the color, so if the three of them are 255 then the pixel is the white color and if the three are 0 than the pixel assumes the color black.

Figure 1. Normal image with ideal conditions.

Figure 2. Normal image with non-ideal conditions.

Fig. 1 represents an image in ideal conditions and the Fig. 2 shows how the ambient alterations make the pixels very different, so it's harder to identify these pixels with the same parameters.

The values observed in some of the pixels makes the range between them so big that when we try to identify the field with the same parameters it's not possible to do so with safety.

The function was based on Akash[3] project, that normalize the image reading the three RGB values and making the following calculations to highlight only the three main colors while making harder to distinguish mixtures of colors.

$$R' = \left(\frac{R}{R+G+B}\right) * 255 \quad (1)$$

$$B' = \left(\frac{B}{R+G+B}\right) * 255 \quad (2)$$

$$G' = \left(\frac{G}{R+G+B}\right) * 255 \quad (3)$$

The R, G and B variables are the actual values for the pixel and the R', B' and G' are the new values. The equations (1), (2) and (3) cause the values to be easily identified as they diminish the difference between the values of each main color when there are variations in the light.

So, by using the equations in each pixel and using the openCV functions to replace RGB by R'G'B' respectively, we can highlight the red, blue and green colors of the image, and minimize the errors we get from any external factor that changes the color shades as demonstrated fig.3 and fig.4.

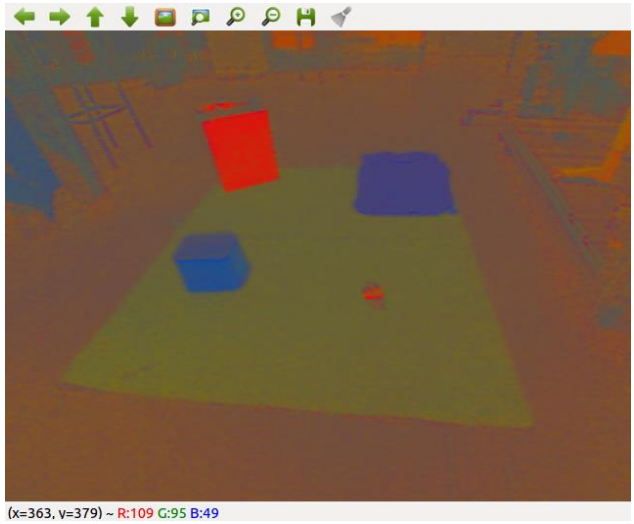

Figure 3. Normalizer on with ideal conditions.

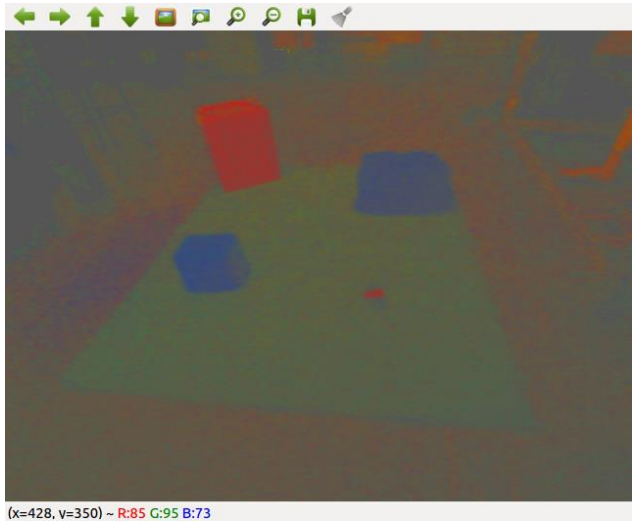

Figure 4. Normalizer on with non-ideal conditions

III. DEVELOPMENT OF THE PROGRAM

To identify the field, the program needs another program to work. One of them define the parameters and writes it in a .xml file. The other use these to identify the field, both of them uses the function mentioned above.

The program that defines the parameters is used for calibration, so it is used once and then the behavior of the robot uses only the second program who reads the results of the first one.

A. Calibration program

The calibration program uses the OpenCV function inRange, this function takes 6 parameters to work: a minimum and a maximum value for each main color, the difference between the minimum and maximum value is called range of each color.

This function takes these parameters and one image and make all the pixels that have all the 3 main colors values inside the range white and all the other pixels, that have at least one parameter out of range, black.

Beyond that the program also uses functions to erode and dilate the pixels of the black and white image to minimize the errors.

The program opens four windows, the normal image (first image), the image normalized (second image), the white and black image (third image) and the track bars.

The 8 track bars are used to define the parameters. They work by ranging the first 6 ones between 0 and 255, and the last two ones between 0 and 25. All of them are used to change the parameters of the OpenCV functions in real time and the minimum ones starts at 0, the maximus ones at 255 and the two last ones in 0.

After generate the 4 windows, the third one is the only one that changes when sliding the track bar.

Figure 5. Normal image with ideal conditions.

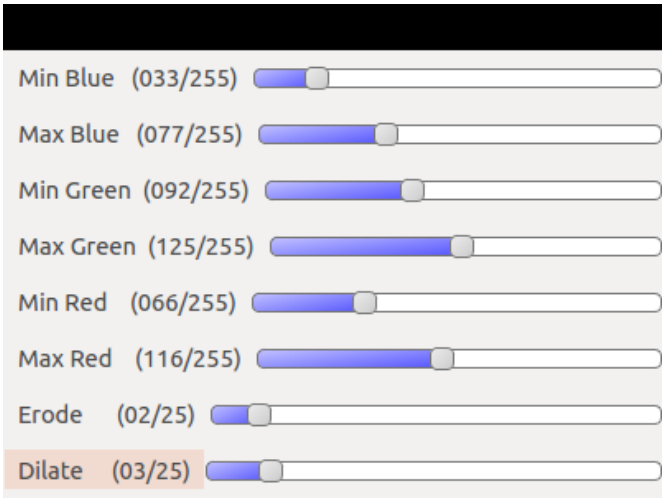

Figure 6. Track bars with defined parameters.

In order to calibrate the image, the program applies the equations (1), (2) and (3) to normalize the image in the first image and store it in the second image. Then it applies the inRange function in this image to generate the third image.

Initially the third image is completely white because all the pixels are inside the range initially defined. Then the user slides the track bar until the third image turns in white only the pixels with the same coordinate that the green ones in the first image (see fig.5, fig.6 and fig.7). After that the user changes the conditions and see if the white pixels in the third image have the same coordinate that the green ones in the first image (see fig.6, fig.8 and fig.9), if not, the user adjust again the parameters in the track bars and keep doing it until the parameters work for different conditions.

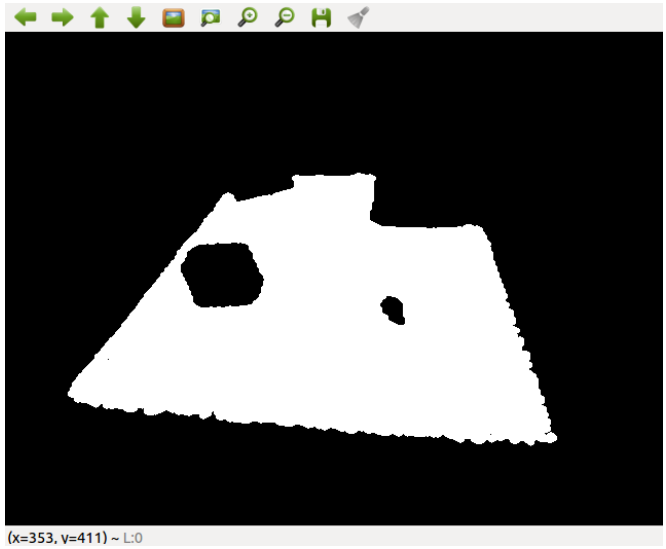

Figure 7. Third image with parameters defined by fig. 5 and fig.6.

Figure 8. Normal image with non-ideal conditions.

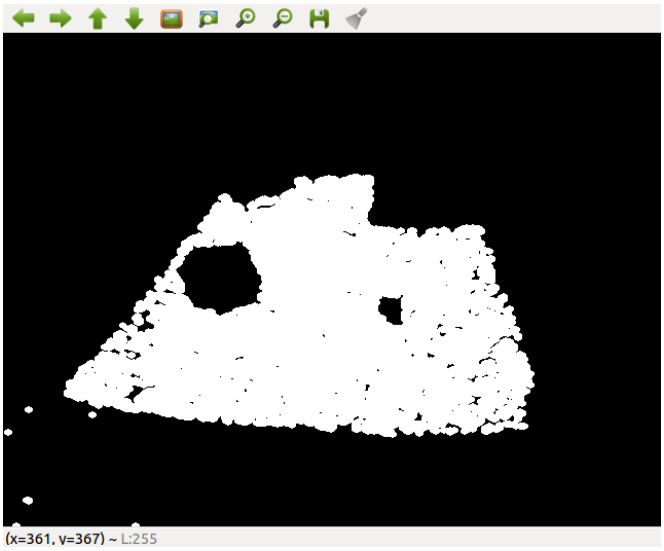

Figure 9. Third image with parameters defined by fig.8 and fig.6.

After obtained the correct parameters the user presses a pre-defined key and then the program writes the parameters in a .xml file and the program end with the calibration process.

B. Identifying the field

After using the calibration program and generating the .xml file the program that the robot uses to identify the field can now be used.

An important aspect of that program is that it doesn't work if the .xml file doesn't exist. So it is needed to use the calibration program in order to the main program work.

The main program doesn't open any window, it just reads the camera's image, apply the function to normalize it and then apply the function inRange to generate the black and white image (see fig.7 and fig.9).

Keep in mind that the program accesses those imagens by the way of variables, the windows are just a way to show it to the user, so it's not needed.

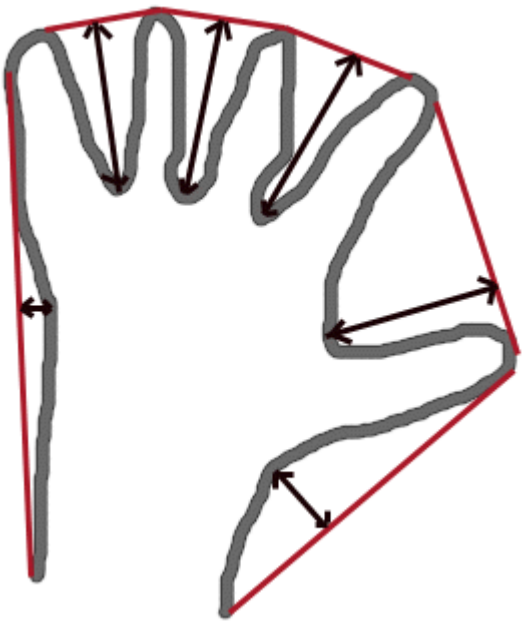

Figure 10. The effect of the function convexHull[4].

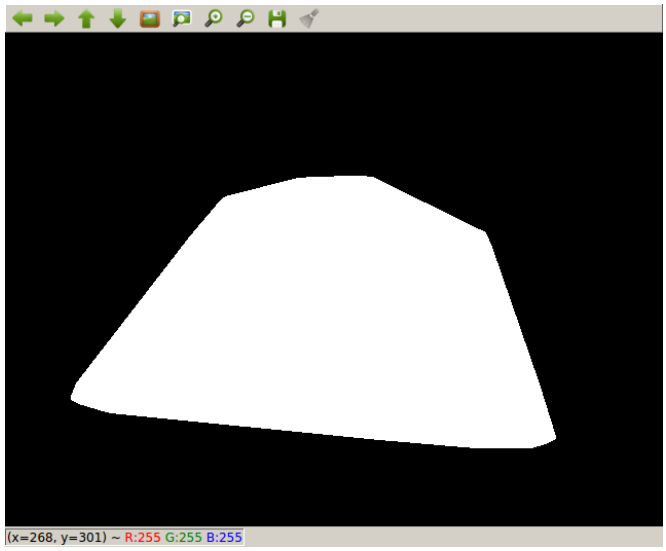

Figure 11. The convexHull function applied in fig.9 and fig.7.

With that image generated (see fig.7 and fig.9) the program makes contours in the white areas. In these contours can be found errors and in order to minimize those errors the program uses the OpenCV function convexHull (see fig.10). It makes the contours became more closely to lines.

After done these operations the program calculates the value for the white areas and then it picks the larger one, so it ignores from the image the other areas.

This is done so a person in green shirt passing in front of the robot isn't recognized as the field.

After this the program keeps returning the image updated image in real time.

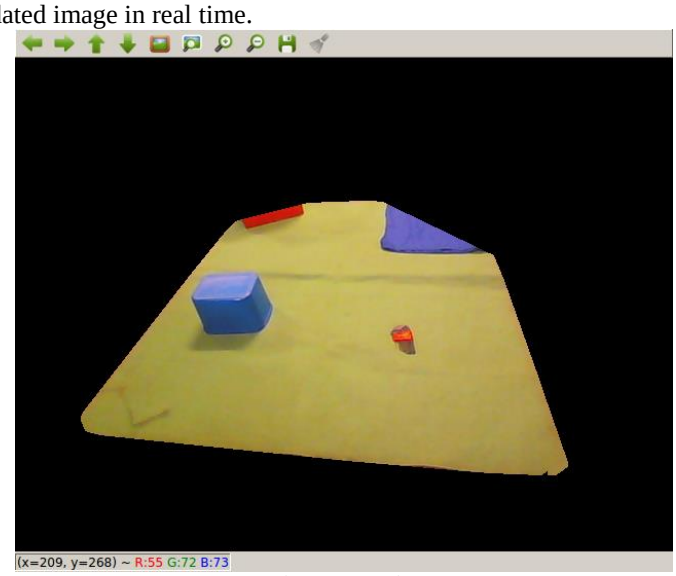

Figure 12. The final vision of the robot.

At this point the program can successfully identify the field, so when the robot behavior tries to identify anything, like the ball or another robot, it can be limited to search these things only in the coordinates that are white in the returning image the function returns.

So the program basically limits the robots vision to the field (see fig. 12).

IV. Conclusion

The main goal with this project was to create a program that could recognize the soccer field of Robocup Humanoid out of the whole image. While developing that we found ourselves facing problems with different light ambient that could cause some errors, since we were finding the field using color recognition, a different shade of the same color could give us some bad data and that is why we did research on a way to avoid those errors.

What we developed here can also be used, for example, on autonomous cars that have to recognize the traffic lights and stop signs since it highlights red and green and it can give a more securely result of the color in different light conditions.

Acknowledgment

The authors thank all the supporters and sponsors. This work was supported by the FEMEC/UFU, FAPEMIG, and it was sponsored by Radix, Central Máquinas and CNPq.

References

[1] RoboCup. Available: http://www.robocup.org/
[2] OpenCV. Available: http://opencv.org/
[3] Akash project: http://akash0x53.github.io/blog/2013/04/29/RGB-Normalization/
[4] OpenCV ConvexHull function: http://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html?highlight=convexhull#convexhull