Design of a video processing algorithm for detection of a soccer ball with arbitrary color pattern

R. Woering

DCT 2009.017

Traineeship report

Coach(es): Ir. A.J. den Hamer

Supervisor: prof.dr.ir. M. Steinbuch

Technische Universiteit Eindhoven
Department Mechanical Engineering
Dynamics and Control Technology Group

Eindhoven, March, 2009

Contents

1 Introduction

This research is performed within the RoboCup project at the TU/e. RoboCup is an international joint project to promote A.I. (Artificial Intelligence), robotics and related fields. The idea is to perform research in the field of autonomous robots that play football by adapted FIFA rules. The goal is to play with humanoid robots against the world champion football of 2050 and hopefully win. Every year new challenges are set to force research and development to make it possible to play against humans in 2050.

An autonomous mobile robot is a robot that is provided with the ability to take decisions on its own without interference of humans and work in a nondeterministic environment. A very important part of the development of autonomous robots is the real-time video processing, which is used to recognize the object in its surroundings. Our robots are called TURTLE's (Tech United RoboCup Team Limited Edition), from now on called Turtle's. On top of the Turtle, a camera is placed to be able to see the world around it. The camera is pointing towards a parabolic mirror, thus creating a 360° view of its surroundings, so called omni-vision. Using this image, the robot is able to determine its own position and find back the obstacles and an orange ball [1]. Object recognition is done by video processing based on color: the ball is red, objects and robots are black, the field is green and the goals were yellow or blue (depending on the side you play on).

Shape recognition is a new step in the RoboCup vision processing because the aim is to play with any arbitrary colored football, colored team opponents and much more. This report is about the design of an algorithm to find an arbitrary colored FIFA football for the RoboCup challenge of this year. The formal problem definition can be described as:

The challenge is to design an algorithm that can find an arbitrary colored foodball based on the captured image from the omni-vision. The used algorithm is a Circle Hough Transform which will be explained in detail in this report.

The exact description of the challenge is as follows [2]: "The challenge is to play at the world championship in China 2008. This challenge is carried out with three different standard FIFA balls. A robot is placed on the field and the ball is placed in front of the robot for 5 seconds. Afterwards the ball is placed at an arbitrary position on the field. The robot now has 60 seconds to find the ball and to dribble it into a predefined goal". The aim of this challenge is to encourage teams to improve their vision routines and to divide the teams in poules for the tournament [3].

In the first chapter a brief overview is given of the found literature. In the second chapter basic elements of image processing is explained like, where an image is build of and how it can be manipulated with filters. The third chapter explains the algorithm of the Circle Hough Transformation (CHT) and how it can detect circles. In the fourth chapter first results will be discussed using $Matlab^{®}$ on images that were taken from the Turtle's. The fifth chapter explains implementation of the real-time CHT algortihm using OpenCV in the Turtle Vision scheme, and the results of those experiments. For the real-time processing the OpenCV libraries are used in combination with Matlab simulink. OpenCV is a computer vision library originally developed by Intel [4]. It focuses mainly on real-time image processing and therefore assumed to be reliable and stable to use for our real-time image processing to detect a soccer ball with arbitrary color pattern. And finally in the last chapter the conclusions and recommendations of using Circle Hough Transformation for real-time arbitrary ball detection on the Turtle's.

More information can be found on http://www.techunited.nl and http://www.robocup.org

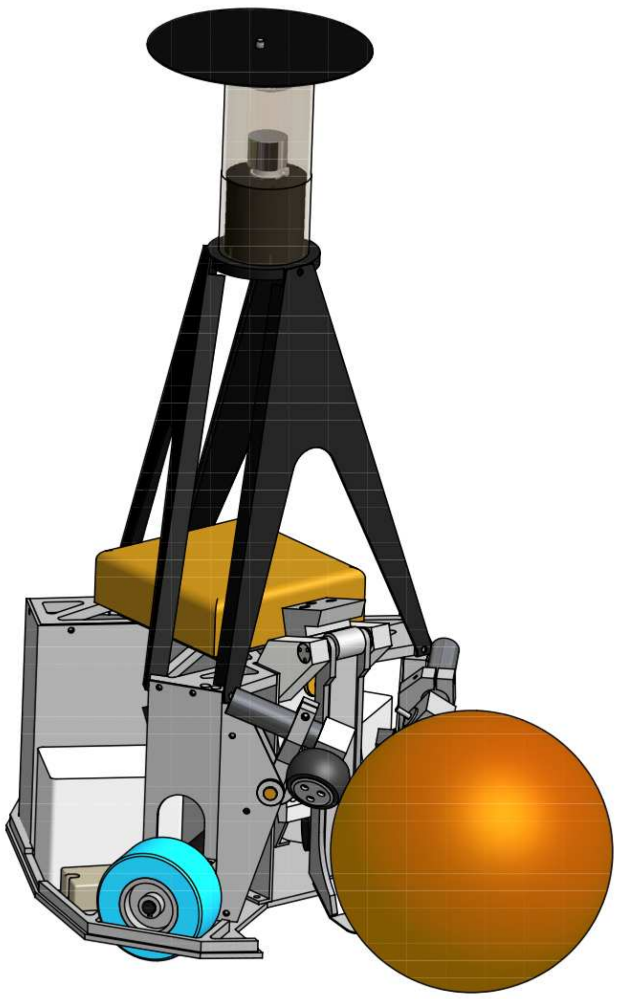


Figure 1: The TURTLE of Techunited

2 Literature

In order to start the project, a small literature study is performed on image processing. Two methods were found for implementing a circle detection. The first using genetic algorithms [11] and the second being a Circle Hough Transformation. Unfortunately there is almost no knowledge on genetic algorithms in our department and not enough time for a study on it because this is a 6 weeks project. The most commonly used method found in papers is the Circle Hough detection algorithm [15]. According to the work presented in [16], the Circle Hough detection should be capable of tracking a football during a football match. Further more the Circle Hough detection shows to be robust to find a circle even when the circle is only partially visable [13]. The decision is made to use the Circle Hough detection related on:

- Relative easy algorithm
- Large information available
- Size invariant possible for close and far away circle detection [10]
- Easy implementation, with OpenCV library

In oder to apply Circle Hough Transformation, a filtered image is required to highlight edges of the objects. Therefore some knowledge of basic image processing is needed. To understand basic image processing techniques information is gathered from books [12] [14] that explain filters and other digital image processing techniques. A lot of information on image processing algorithms like filters are also found on the internet [5] which is also used in the chapter about basics imaging processing.

3 Basics of image processing

In this chapter some basic information is explained about image processing. Color spaces are discussed, which are used to represent the images. A brief explanation of different filters that can be applied on images. Furthermore the edging technique is highlighted which is used to edge the image for using the circle detection. Figure 2 depicts the main steps which will be described in this chapter.

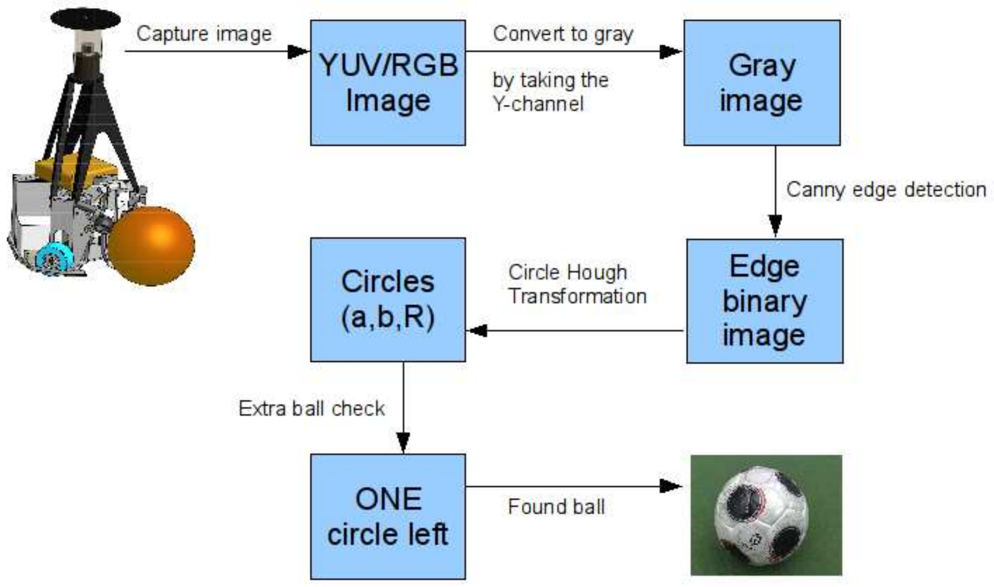


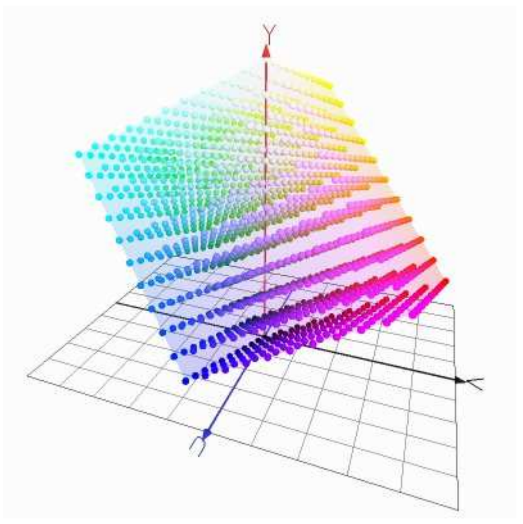
Figure 2: An overview of the report, and how the arbitrary colored ball detection works

3.1 YUV and RGB color spaces

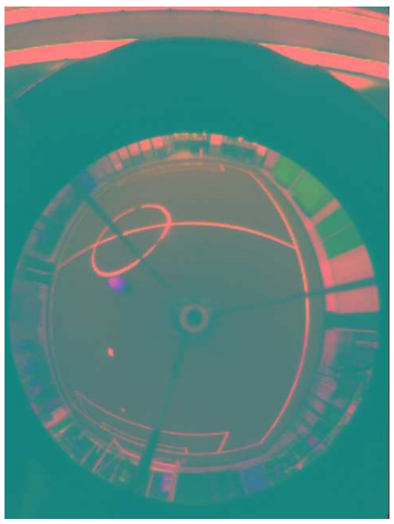
A color image can be described as an array of bytes, read from the camera. The matrix captured from the Turtle (lxbx3) is 640x480x3 with 640 being the length of the image, 480 the width and depth 3 for the three color indices of every pixel. The matrix read from the camera is in an YUV configuration, which is a standard color space that originally comes from analogue television transmission. Y is linked to the component of luminance, and U, V are the chrominance components. In Figure 3(a) a color space model of the YUV is given to give a better overview of its structure. In Figure 3(b) an image of the Turtle's view is given. This is the actual YUV image interpreted as a RGB image that the Turtle "sees".

We will first use the RGB color space (Red Green Blue) to test the circle detection software. The RGB is not more then a transformation from the YUV space given by Equation 1. The reason for using the RGB color space for testing instead of the YUV is that in the RGB space the Red ball is fully separable from the background based on the R color channel. The Green "grass" is then visable on the G channel of the RGB space. Because the Green and Red are independent in the RGB space it is easier to test the filters on the red ball in the R channel instead of the YUV space where they are mixed in the YUV channels. The RGB space is shown in Figure 4. By using the R channel for finding a Red ball helps to test the edge detection and high-pass filter which are explained in next paragraph.

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1.140 \\ 1 & -0.395 & -0.581 \\ 1 & 2.032 & 0 \end{bmatrix} \begin{bmatrix} Y \\ U \\ V \end{bmatrix} \tag{1}$$

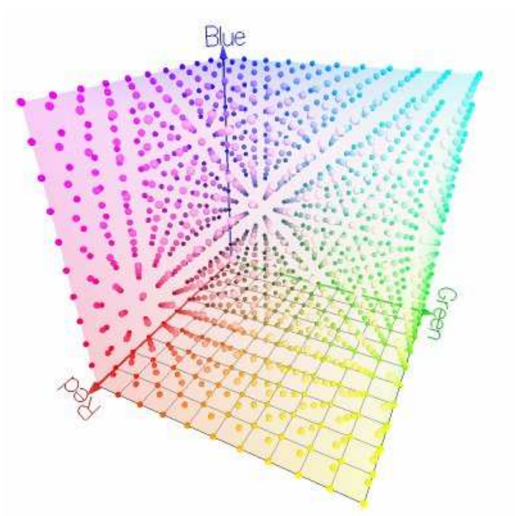


(a) YUV color space

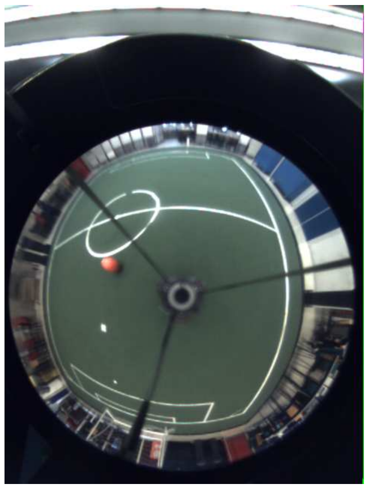


(b) YUV color space as seen by the turtle

Figure 3: YUV color space overview



(a) RGB color space



(b) RGB image from the turtle after conversion from YUV

Figure 4: RGB color space overview

The YUV color space has the advantage of the Y channel which gives a very good gray plot of the image where RGB has its practical use for testing on a red ball on a green field. In appendix A an image is shown with all separate channels of the YUV and the RGB image.

3.2 Linear Filters

In this section, we give a brief summary of the functionality and reason for using filters. Image filtering allows you to apply various effects on photos. The trick of image filtering is that you have a 2D filter matrix, and the 2D image that are convoluted with eachother. Convolution is a simple mathematical operation which is fundamental to many common image processing operators Equation 2. Convolution provides a way of 'multiplying together' two arrays of numbers, generally of different sizes, but of the same dimensionality, to produce a third array of numbers of the same dimensionality. This can be used in image processing to implement operators whose output pixel values (depending on kernel, high-pass or low-pass characteristics) are simple linear combinations of certain input pixel values. The 2D Convolution block computes the two-dimensional convolution of two input matrices. Assume that matrix A has dimensions (Ma, Na) and matrix B has dimensions (Mb, Nb) [6]. When the block calculates the full output size, the equation for the 2-D discrete convolution is given by Equation 2.

$$C(i,j) = \sum_{m=0}^{(Ma-1)} \sum_{n=0}^{(Na-1)} A(m,n) * B(i-m, j-n) \tag{2}$$

Where $0 \leqslant i < Ma + Mb - 1$ and $0 \leqslant j < Na + Nb - 1$.
Many useful image processing operations may be implemented by filtering the image with a selected filter. Digital Image Processing defines a large number of smoothing, sharpening, noise reduction, and edge filters. Additional filters may be easily added or designed using the filter design functionality of the package. Box filter, Gaussian filter, and Smoothing filter are all variants of so-called smoothing filters. They produce a response that is a local (weighted) average of the samples of a signal [7].

To get an optimal result for edge detection of the image some filters are tested to make color edges smoother or more blurred. The filter that were examined are those who are available in Matlab® using $fspecial$. The basic filter available are:

- 'Average' averaging filter
- 'Disk' circular averaging filter
- 'Gaussian' Gaussian lowpass filter
- 'Laplacian' filter approximating the 2D Laplacian operator
- 'LoG' Laplacian of Gaussian filter
- 'Motion' motion filter
- 'Prewitt' Prewitt horizontal edge-emphasizing filter
- 'Sobel' Sobel horizontal edge-emphasizing filter
- 'Unsharp' unsharp contrast enhancement filter

For more mathematical information and details how each of these filters work can be found in book [14]. A couple of filters are tested and the result is shown in Figure 5. The choice for which filter to use is done by testing. The result is that the averaging filter, Gaussian low-pass filter and Laplacian of Gaussian filter have the most potential and are therefor use in the Matlab script of filtering the image.

Averaging filtering

The idea of averaging filtering is simply to replace each pixel value in an image with the averaging ('mean') value of its neighbors, including itself. This has the effect of eliminating pixel values which are unrepresentative of their surroundings. Mean filtering is usually thought of as a convolution filter. Like other convolutions it is based around a kernel, which represents the shape and size of the neighborhood to be sampled when calculating the mean. Often a 3x3 square kernel is used, as shown in Equation 3 below.

$$\begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix} \tag{3}$$

Gaussian low-pass filter

The Gaussian smoothing operator is a 2D convolution operator that is used to 'blur' images and remove detail and noise. In this sense it is similar to the mean filter, but it uses a different kernel that represents the shape of a Gaussian ('bell-shaped') hump. In 2D, an isotropic (i.e. circularly symmetric) Gaussian has the form:

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{4}$$

where σ is the standard deviation of the distribution. The idea of Gaussian smoothing is to use this 2D distribution as a 'point-spread' function, and this is achieved by convolution. Since the image is stored as a collection of discrete pixels we need to produce a discrete approximation to the Gaussian function before we can perform the convolution. In theory, the Gaussian distribution is non-zero everywhere, which would require an infinitely large convolution kernel, but in practice it is effectively zero more than about three standard deviations from the mean, and so we can truncate the kernel at this point. Equation 5 shows a suitable integer-valued convolution kernel that approximates a Gaussian with a σ of 1.0.

$$\frac{1}{273} \begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 26 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 26 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix} \tag{5}$$

Laplacian of Gaussian filter (LoG)

The Laplacian is a 2D isotropic measure of the 2nd spatial derivative of an image [14]. The Laplacian of an image highlights regions of rapid intensity change and is therefore often used for edge detection. The Laplacian is often applied to an image that has first been smoothed with something approximating a Gaussian smoothing filter in order to reduce it's sensitivity to noise, and hence the two variants will be described together here. The operator normally takes a single graylevel image as input and produces another graylevel image as output.

The Laplacian $L(x, y)$ of an image with pixel intensity values $I(x, y)$ is given by:

$$L(x, y) = \frac{\delta^2 I}{\delta x^2} + \frac{\delta^2 I}{\delta y^2} \tag{6}$$

Since the input image is represented as a set of discrete pixels, we have to find a discrete convolution kernel that can approximate the second derivatives in the definition of the Laplacian. Two commonly used small kernels are shown in Equation 7. Using one of these kernels, the Laplacian can be calculated using standard convolution methods.

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \tag{7}$$

Because these kernels are approximating a second derivative measurement on the image, they are very sensitive to noise. To counter this, the image is often Gaussian smoothed before applying the Laplacian filter. This preprocessing step reduces the high frequency noise components prior to the differentiation step. This can also be combined in one operation.

The 2D LoG function centered on zero and with Gaussian standard deviation σ has the form:

$$L(x,y) = \frac{1}{\pi\sigma^3}\left[1 - \frac{x^2+y^2}{2\sigma^2}\right] e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{8}$$

Unsharp filter

The unsharp filter is a simple sharpening operator which derives its name from the fact that it enhances edges (and other high frequency components in an image) via a procedure which subtracts an unsharp, or smoothed, version of an image from the original image. The unsharp filtering technique is commonly used in the photographic and printing industries for crispening edges.

Unsharp masking produces an edge image $G(x,y)$ from an input image $F(x,y)$ via

$$G(x,y) = F(x,y) - F_{smooth}(x,y) \tag{9}$$

where $F_{smooth}(x,y)$ is the smoothed version of $F(x,y)$.

3.3 Edge detection

The edge detection is the most crucial part of object recognition which is based on shape recognition, it converts a color image to a binary image by using a combination of high-pass filters and truncation. It has the aim to identify color borders in a digital image at which the image color or brightness changes rapidly. There are several methods to find edges which are listed below, they differ in combination of high pass filter and truncation. The outcome of an edge detection is a binary image (Black and White) with on the edges a 1 or so called white spot. The edge detection watches the change in color between neighbor pixels and uses a threshold to define if the distance of the color between pixels is larger then the threshold to mark it as an edge. The most known edge detection is the one with uses a canny filter [14], this is also supported in Matlab®. In total Matlab® has six different edge detections algorithms available.

- Sobel Method
- Prewitt Method
- Roberts Method
- Laplacian of Gaussian Method
- Canny Method
- Class Support

The filter used for edge detection is the canny filter and the sobel, but it showed after testing that the canny filter was the most suitable for clear edges and the easiest to use. Figure 6 shows the result of the canny edge detection on the R channel of the RGB images as shown in Figure 4(b). The image is now ready to perform a circle-ball recognition algorithm as will be explained in the next chapter. In Appendix B the canny edge detections are shown of the other filtered images that are shown in Figure 5. The reason of using other filters on the original image and not only the original images is that the ball was not always found with a canny edge detection an on the original image. But this will be explained in the next chapters.

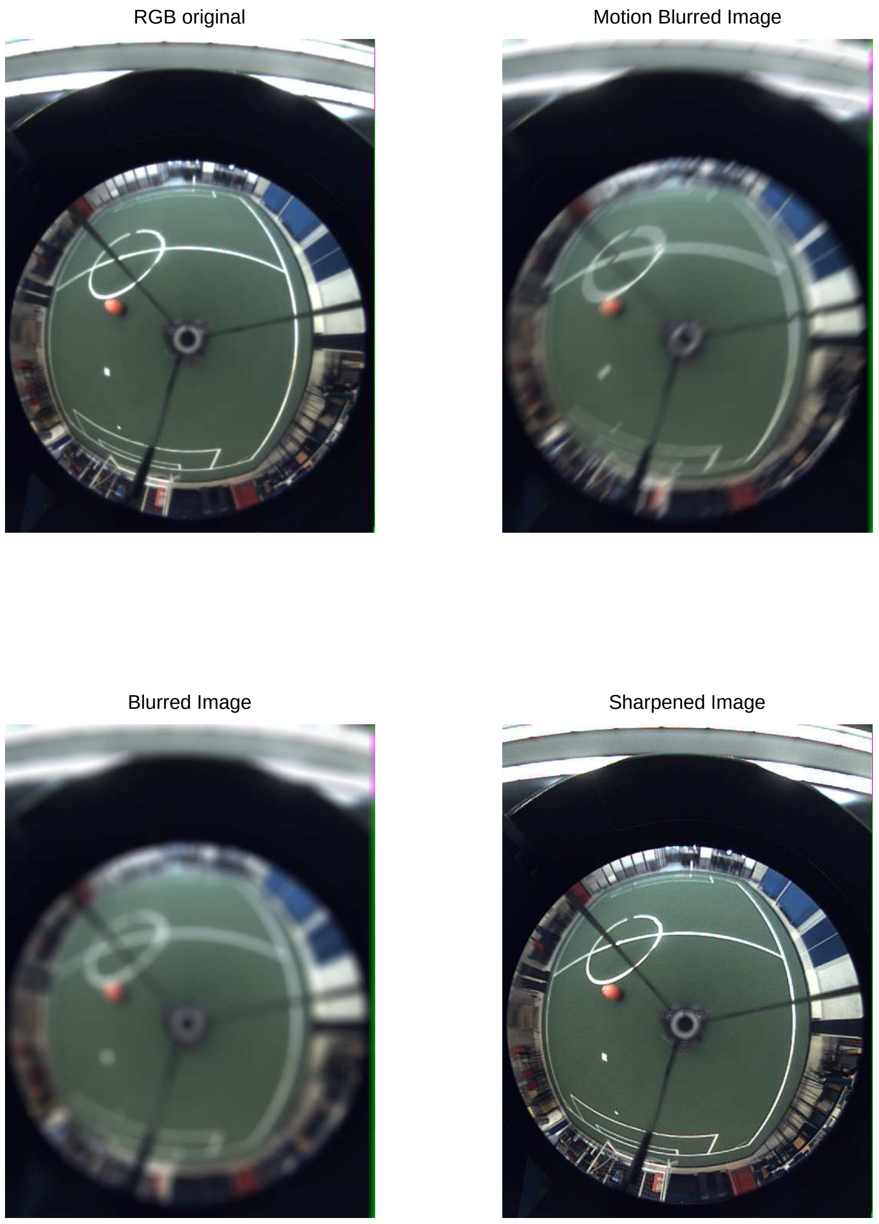


Figure 5: Left top the original image is shown, Right top shows the same image but using a motion filter ('motion',20,24), Left bottom is the disk filter ('disk',10) and right buttom the sharpened image ('unsharpened')

Canny method

The Canny operator was designed to be an optimal edge detector. It takes as input a gray scale image, and produces as output an image showing the positions of tracked intensity discontinuities.

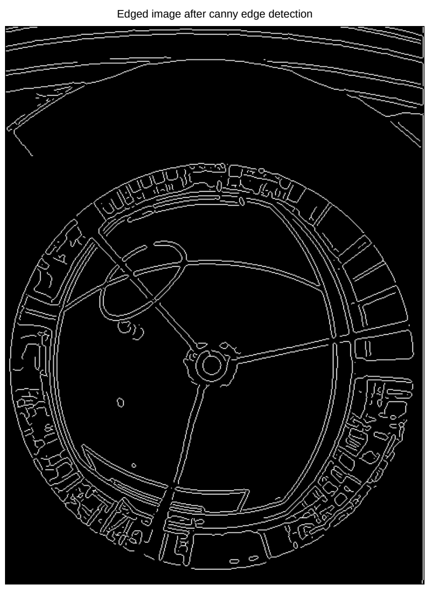


Figure 6: The result of the standard canny edge detection of the gray converted images

The Canny operator works in a multi-stage process. First of all the image is smoothed by Gaussian convolution. Then a simple 2D first derivative operator is applied to the smoothed image to highlight regions of the image with high first spatial derivatives. Edges give rise to ridges in the gradient magnitude image. The algorithm then tracks along the top of these ridges and sets to zero all pixels that are not actually on the ridge top so as to give a thin line in the output, a process known as non-maximal suppression. The tracking process exhibits hysteresis controlled by two thresholds: T1 and T2, with T1 > T2. Tracking can only begin at a point on a ridge higher than T1. Tracking then continues in both directions out from that point until the height of the ridge falls below T2. This hysteresis helps to ensure that noisy edges are not broken up into multiple edge fragments.

Sobel method

The Sobel operator performs a 2D spatial gradient measurement on an image and so emphasizes regions of high spatial frequency that correspond to edges. Typically it is used to find the approximate absolute gradient magnitude at each point in an input grayscale image.

In theory at least, the operator consists of a pair of 3x3 convolution kernels as shown in Table below. One kernel is simply the other rotated by 90 degrees.

$$Gx = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ 1 & 0 & 1 \end{bmatrix} Gy = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{10}$$

These kernels are designed to respond maximally to edges running vertically and horizontally relative to the pixel grid, one kernel for each of the two perpendicular orientations. The kernels can be applied separately to the input image, to produce separate measurements of the gradient component in each orientation (call these Gx and Gy). These can then be combined together to find the absolute magnitude of the gradient at each point and the orientation of that gradient.

4 Circle Hough Transform (CHT)

To find a ball by shape, the Circle Hough Transform is used [17]. The Hough transform can be used to determine the parameters of a circle out of a large number of points that are located on the contour of the circle. To perform this operation, an edged image is used as already explained in the previous chapter. If an image contains many points, some of which fall on the contours of circles, then the job of the search program is to find the center of the ball described by the parameter (a, b) for each circle with a fixed radius. A circle with radius R and center (a, b) can be described with the parameterization given in Equations 11. The basic idea of CHT is that on the detected edge pixels a circle is drawn of the searched circle radius. On each edge pixel a circle is drawn and those drawn circle are put in a matrix the so called accumulator array and is raised with 1. The values in this matrix are compared to a preset threshold, those values higher then the threshold are the centres of fixed radius circles. A graphical overview is shown in Figure 7

$$\begin{aligned} x &= a + R\cos(\theta) \\ y &= b + R\sin(\theta) \end{aligned} \tag{11}$$

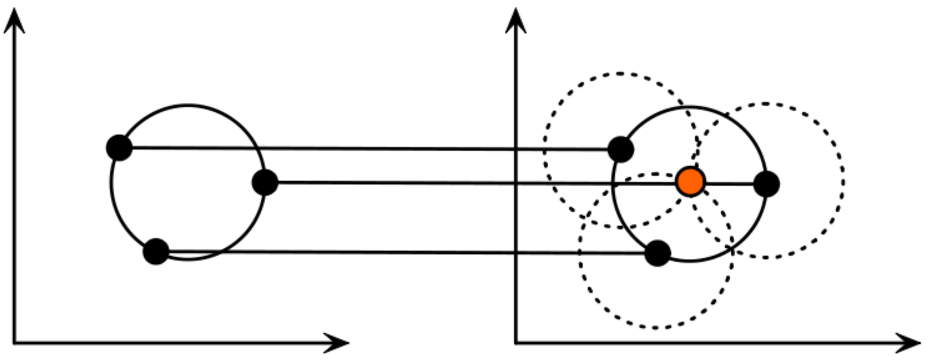


Figure 7: Each point in geometric space (left) generates a circle in parametric space (right). The circles in parameter space intersect at the (a, b) that is the center in geometric space.

At each edge point a circle is drawn with the center in the point with the desired radius. This circle is drawn in the parameter space, such that our x axis is the *a* value and the y axis is the *b* value, while the z axis is the radii. At the coordinates which belong to the perimeter of the drawn circle we increment the value in our accumulator matrix which essentially has the same size as the parameter space. In this way we sweep over every edge point in the input image drawing circles with the desired radii and incrementing the values in our accumulator. When every edge point and every desired radius is used, we can turn our attention to the accumulator. The accumulator will now contain numbers corresponding to the number of circles passing through the individual coordinates. The accumulator is then set to a threshold number, at every coordinate that is larger then the threshold is a possible ball (circle) detected. In Figure 8 the edged image is shown and next to it the accumulator of that image after CHT transformation. The locus of (a, b) points in the parameter space fall on a circle of radius R centered at (x, y). The true center point will be common to all parameter circles, and can be found at the maximum values in the Hough accumulation array.

4.1 Extra ball check

Figure 8 depicts the accumulator matrix of a captured image. As can be seen in this image, the image is very chaotic with all the circle drawings. The ball diameter is set to 10 pixels and threshold to 16 after some testing, because the diameter of the ball in the detection region (0,3 - 5 meters) is 8 to 15 pixels wide. The threshold of 16 is found by testing. Changing the value

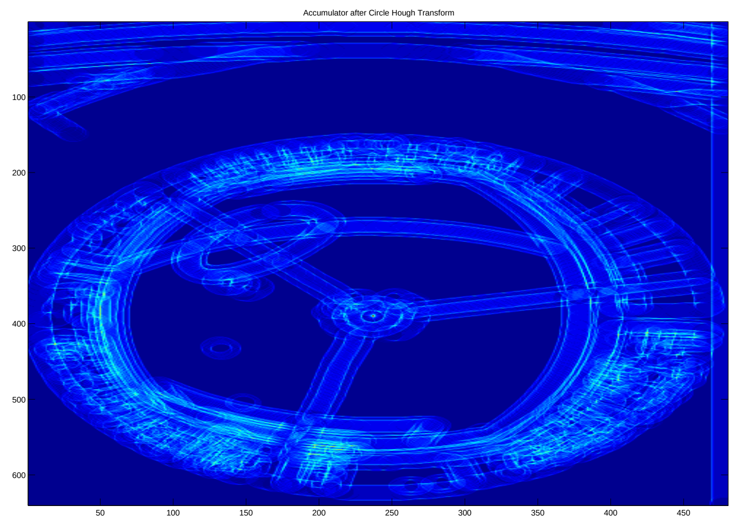


Figure 8: The accumulator after CHT on the original RGB canny edged image

lower then 16 increases the amount of false detected circles and 16 means that at least half of the contour of the circle is edged. The threshold of 16 is still very low, but reduces the amount found circles to about 20. This has as result that many points have high accumulator values and are incorrectly indicated as a ball center. These incorrectly detected circles are mostly corners of the fieldlines since they contain many points that appear approximately in the shape of a half circle. To remove these points an additional check is implemented. This extra check is implemented to check if the detected circle coordinate is really a ball. Based upon the shadow at the bottom of the ball and the light at the top, this extra ball feature can check whether it is a ball. For example fieldlines don't cause shadows. This extra check will look at all possible found ball centers and see if they have that ball feature. This is done by making a line from the center of the possible ball to the center of the image. The center of the picture is the robot itself in the middle of the 360 degree view. An overview of the extra ball check is shown in Figure 9 where R is the center of the possible found ball, 0 is the center of the Turtle in the photo and L the line where the check is performed on.

1. Check if the coordinates lie between 5 and 180 pixels from the center O.
2. Check the Y-channel over the line L.

$$\begin{gathered} R(x_R, y_R) \Rightarrow R(r_R, \theta_R) \\ for\ i = 1 : k \\ n = (0 : 1 : 55) \\ L(n) = r_R - 40 + n \\ x_n = (L(n)) \cos \theta_R \\ y_n = (L(n)) \sin \theta_R \\ M(i) = Y(x_n, y_n) \\ end \end{gathered} \tag{12}$$

The first check is to eliminate an infeasible ball location in the signal. This means, the center of the robot could be seen as a circle (it is a circle but of course not the ball). And everything further away that 180 pixels from the center is to small to determine for sure that it is a ball with a good certainty.

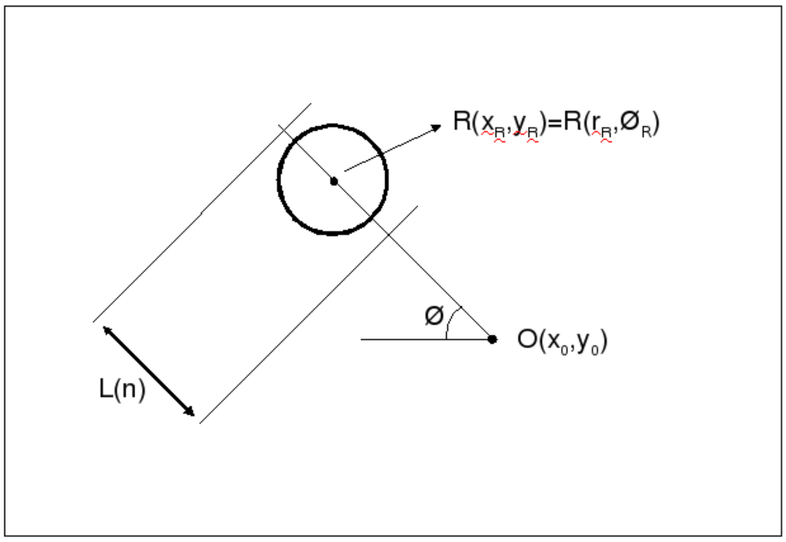


Figure 9: An image to show the working of the extra ball check.

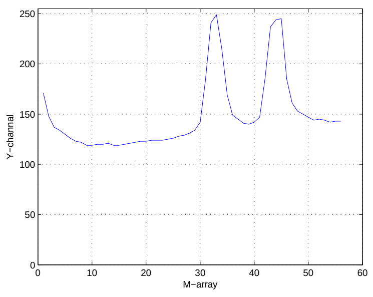
The second check is the check from the radius from the center to the coordinates of a possible ball position. If $O = (x_0, y_0)$ is the center of the robot, and $R = (x_R, y_R)$ is the possible ball location a virtual line is taken between those points. First make polar coordinates from the Y-channel with O being the center by Equation 13. The next step is to make an array L this is a line through the point $R(r_R, \theta_R)$ starting from the origin $O = (x_0, y_0)$. The values from $(-40r < (r_R, \theta_R) < +15r, \theta) = [L]$ are saved in [L] array. Next the corresponding Y-values of [L] coordinates are put in the array [M] and this results into the following Figures 10 or 11. The algorithm is described in 12 with k the number of found possible balls.

To obtain θ in the interval [0, 2π), the matlab command *cart2pol* is used to get the θ and Radius (R) of the found center of the possible ball position.

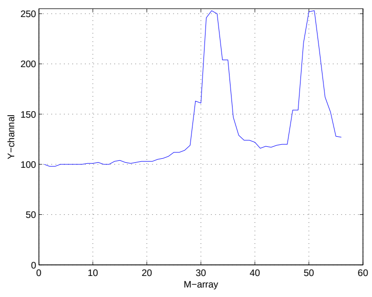
$$r = \sqrt{(x_R - x_o)^2 + (y_R - y_0)^2}$$
$$\theta = \begin{cases} \arctan(\frac{y_R - y_0}{x_R - x_0}) & \text{if } (x_R - x_0) > 0 \text{ and } (y_R - y_0) \geq 0 \\ \arctan(\frac{y_R - y_0}{x_R - x_0}) + 2\pi & \text{if } (x_R - x_0) > 0 \text{ and } (y_R - y_0) < 0 \\ \arctan(\frac{y_R - y_0}{x_R - x_0}) + \pi & \text{if } (x_R - x_0) < 0 \\ \frac{\pi}{2} & \text{if } (x_R - x_0) = 0 \text{ and } (y_R - y_0) > 0 \\ \frac{3\pi}{2} & \text{if } (x_R - x_0) = 0 \text{ and } (y_R - y_0) < 0 \end{cases} \tag{13}$$

Now that we have the M array, the check can be done to check if the possible ball position is really a ball and not something else like a fieldline or Turtle. The check has two states. First state is to determine the “green“ of the grass. So the first 15 points of the M matrix should give a reference of the Y-channel value of the color green. Comparing this to the Y-value just under the ball (around postion M=22) there should be the shade of the ball. So there should be a value off Y below the referenced green. (mostly below the value of 60). The next check is done around M=44, the top of the ball, this value of Y should be above 230 because on top of the ball the light is very bright. By these checks we can be sure that we don't confuse lines Figure 10(a), 10(c) and 10(d) for a ball. Or as can be seen in Figure 10 for other strange objects. In Figure 11 the M matrix is shown of a red ball . Figure 11(a) is the M array of the used image as shown in the previous chapter, while Figure 11(b) is the same ball from a different picture in an ideal situation in the middle of the field with just green around it.

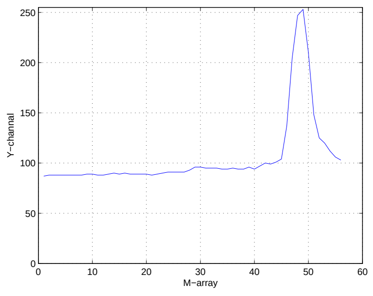
Concluding that the shadow underneath the ball is a very good check to determine if the found circle is a ball or not.



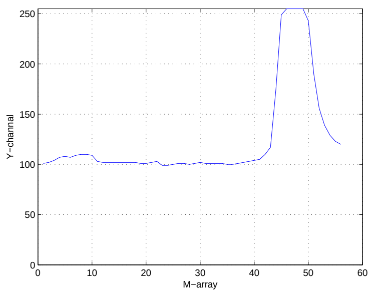
(a) No ball-1



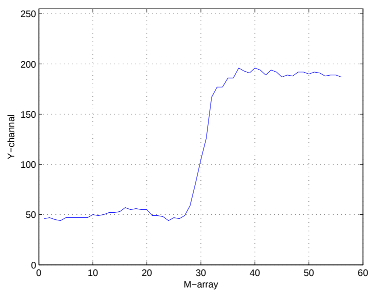
(b) No ball-2



(c) No ball-3

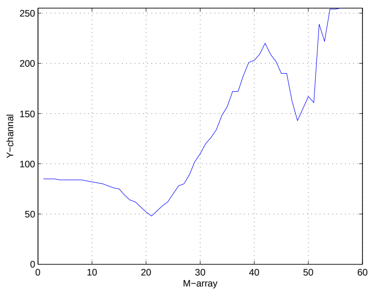


(d) No ball-4

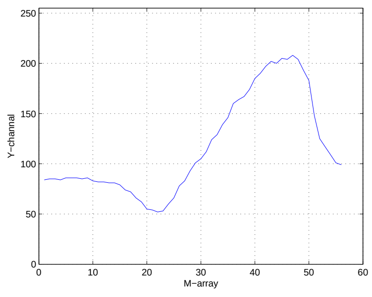


(e) No ball-5

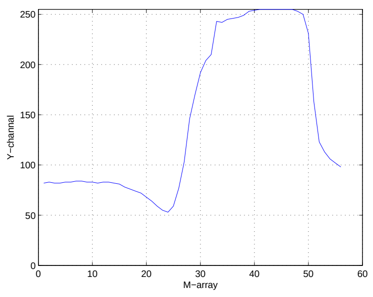
Figure 10: The [M] plot if it is NO ball



(a) the ball-1



(b) the ball-2



(c) the ball-3

Figure 11: The [M] if it is a ball

5 Matlab

In this chapter the theory of previous chapters is combined and put in a chain as depicted in 2. The tests are done on arbitrary colored balls that are shown in Figure 12. Until now the algorithm is only tested on a red ball and it works satisfying in matlab using only captured images from the robot. The algorithm is put to the test because of the different colors, lines, and figures on the ball that will influence the edge detection and therefor also the CHT. Now only the Y-channel of the YUV input image is used to detect any of the following balls.

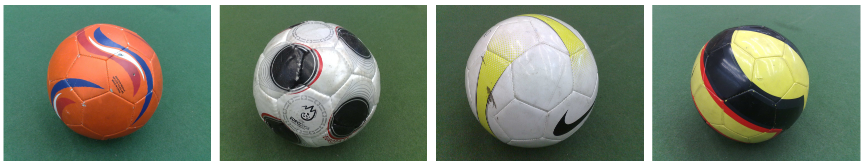
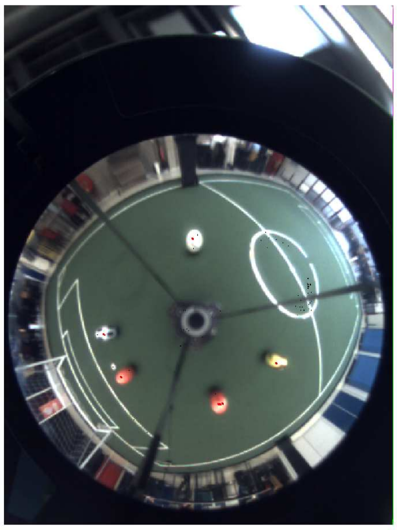

(a) Ball number 1 (b) Ball number 2 (c) Ball number 3 (d) Ball number 4

Figure 12: The different balls to be found by CHT

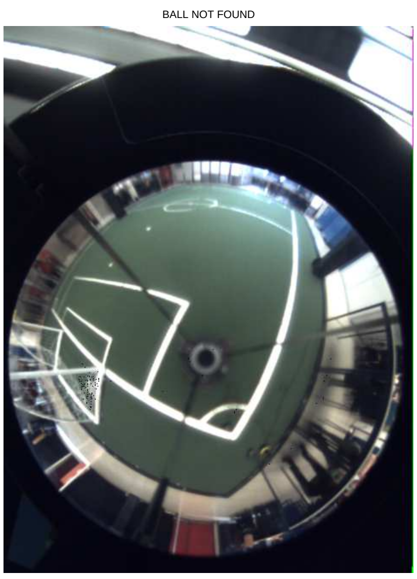
The matlab code is tested by taking a photo with the Turtle and then put on an external PC where the matlab code is started. It should be considered that the parameters like: threshold and shadow spot of the line M are optimized, and normally only one ball can be detected. To demonstrate the result the “one-ball detection” is deleted to show that all the balls can be found in the first images Figure 13. The black dots which can be seen in the center of the field are possible balls but they are eliminated by the second check which is discussed in previous chaptor. The small RED spot on the ball indicates that the first the CHT found a circle (black spot) and secondly the shadow check is positive for this circle. So if both checks are positive we can be positive that it is a ball and this is then marked with the red spot.

The results of the algorithm shown in Figures 14, prove that all balls are found in different positions on the field. Unfortunately in some images which are also shown in Figure 15, points out the biggest problem of the CHT’s that the ball is to small to find a clear circle after edge detection. The pixel diameter of the ball is smaller 8 pixels and that is to low to really find a good and clear circle using different edge detections methods explained in previous chapters. In oder to test the algorithms robustness some pictures are captured from the Turtle but with NO ball in the field, and this gave the result that no ball is found.

By manual optimize the CHT, shadow and light spot in the Matlab code also the balls which where first not found when the ball was just to far away or wrong filter used for edging can be found. But then the code is optimized algorithm is only valid for one picture. The adapted parameters are: threshold and circle radius within the CHT, that the spot of the shadow is very variable by ball distance. In general by adapting the CHT parameters it means that a lot of non-circles are detected as a circle and by chance also some edge pixels around the ball are detected as a circle. Further the second check has to be manual adapted, by replacing the center postion / top and shadow spot if the ball at this large ball distance for each image, because the ball has a smaller diameter. Because of this problem that the ball diameter gets to small the algorithm is only capable of finding a ball in a range of 0,3 - 5 meters.



(a) CHT with the second check finding all the balls, that can be seen by the red spot on the ball

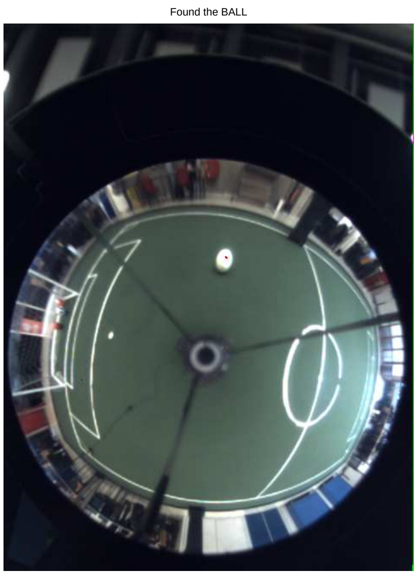


(b) Test image if there is no ball found in a image without a ball

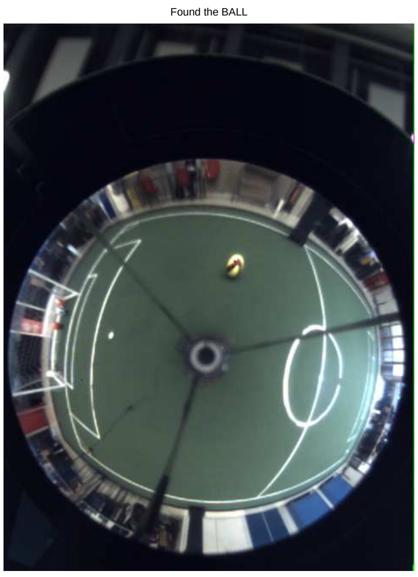
Figure 13: Testing the CHT on every ball separately on the field and an empty picture

(a) Ball number 1

(b) Ball number 2



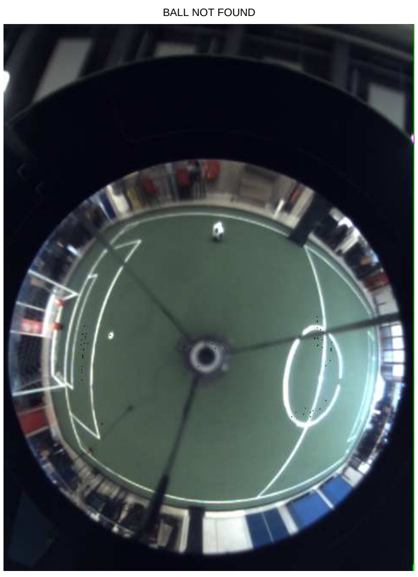
(c) Ball number 3



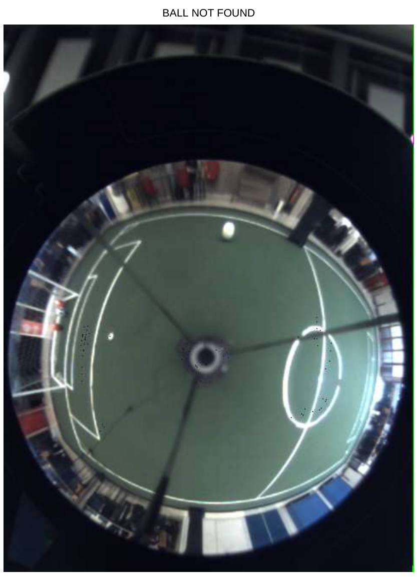
(d) Ball number 4

Figure 14: Testing the CHT on every ball separately on the field

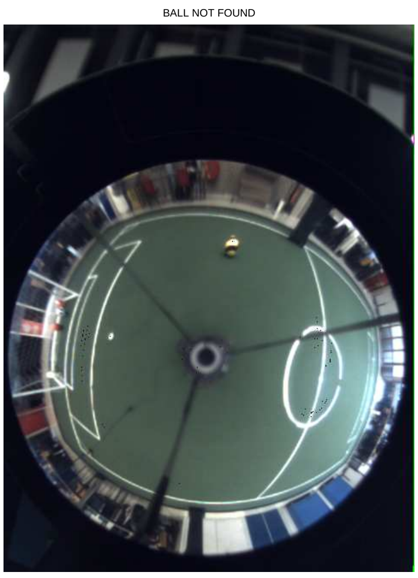
(a) Ball number 1



(b) Ball number 2



(c) Ball number 3



(d) Ball number 4

Figure 15: Testing the CHT on every ball separately on the field

Some conclusion could be made with this matlab simulations. The arbitrary colored ball detection is working on images taken from the Turtle. After applying different filters in most cases the ball was found. We had the most difficulties with detecting the ball when the ball lies on a white line so that there is no clear shadow underneath the ball or if the ball is to far away. By using the second check based on the values of the Y-channel over the ball we at least know that if we find a ball it is a ball and not something else. The second ball check works perfectly by identifying a circle as a ball by its shadow underneath the ball and bright spot on top of the ball. At far distances the radius of the ball gets to small so that the circle diameter where the CHT has to look for finds to many possible balls that it is more a random guess then a defined circle. Another problem by a very small ball diameter is that line L is so short that the shadow spot is not clearly visable.

6 OpenCV

In order to perform the CHT in real-time on the Turtles the code should be implemented in C-code. Matlab is not able to perform image processing in real-time, therefor C-code libraries are implemented with the opensource OpenCV. OpenCV is a computer vision library originally developed by Intel. It is free for commercial and research use under a BSD license. The library is cross-platform, and runs on Windows, Mac OS X, Linux, PSP, VCRT (Real-Time OS on Smart camera) and other embedded devices. It focuses mainly on real-time image processing, as such, if it finds Intel's Integrated Performance Primitives on the system, it will use these commercial optimized routines to accelerate the code [4].

OpenCV is chosen since it contains an implementation of circle detection using hough transformation. The previous knowledge of the hough circle detection is now implemented in C-code and implemented in the vision scheme of the simulink program.

The parameters needed for the cvHoughCircles are:

- Image = the input 8-bit single-channel grayscale image.
- Circle storage = The storage for the circles detected. It can be a memory storage (in this case a sequence of circles is created in the storage and returned by the function) or single row/single column matrix (CvMat*) of type "CV-32FC3", to which the circles' parameters are written. The matrix header is modified by the function so its cols or rows will contain a number of lines detected. If circle-storage is a matrix and the actual number of lines exceeds the matrix size, the maximum possible number of circles is returned. Every circle is encoded as 3 floating-point numbers: center coordinates (x,y) and the radius.
- Method = Currently, the only implemented method is CV-HOUGH-GRADIENT.
- dp = Resolution of the accumulator used to detect centers of the circles. For example, if it is 1, the accumulator will have the same resolution as the input image, if it is 2 - accumulator will have twice smaller width and height, etc.
- Minimal distance = Minimum distance between centers of the detected circles. If the parameter is too small, multiple neighbor circles may be falsely detected in addition to a true one. If it is too large, some circles may be missed.
- Param1 = The first method-specific parameter. In case of CV-HOUGH-GRADIENT it is the higher threshold of the two passed to Canny edge detector (the lower one will be twice smaller).
- Param2 = The second method-specific parameter. In case of CV-HOUGH-GRADIENT it is accumulator threshold at the center detection stage. The smaller it is, the more false circles may be detected. Circles, corresponding to the larger accumulator values, will be returned first.
- Minimum radius = Minimal radius of the circles to search for.
- Maximum radius = Maximal radius of the circles to search for. By default the maximal radius is set to max(image-width, image-height).

It has to be mentioned that the documentation of OpenCV leaks in a lot of ways. There is only a little on-line documentation available on the internet [8] but this not up to date. The input parameters are all explained at [8], but minimum and maximum radius are not documented. These parameters give a search region for which sizes the circle should be scanned for. Unfortunately this does not work satisfactionly. By tests it shows that if a circle with diameter 15 has to be found, the cvHoughCircles gives different outcomes for changing minimum radiuses from 8/9/10 and maximum radiuses 18/19/20. This is a known problem, but the OpenCV is not anymore supported by Intel and therefore not up to date.

In general the code works good on the Turtle's. After implementing also the second check the result was positive on finding the arbitrary balls. And generated images for the Turtle's is shown in Figure 16 but here also for demonstration purpose the one-ball detection is disabled to find as much as possible balls.

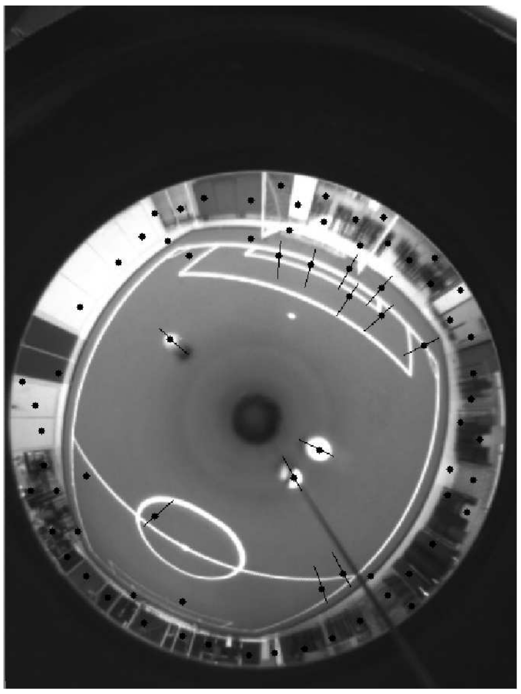


Figure 16: An image taken from the Turtle which was running the OpenCV code and the second check.

Implementing extra filters and choosing different edging techniques as in the matlab code is not tested for the cvHoughCircles because the first results of the cvHoughCircles where very promissing. Some vision toolboxes from simulink where tested, but they need an extreme effort of processor power that it is by far from real-time implementation.

6.1 OpenCV real-time testing

The OpenCV code is tested real-time, but it did not work as good as was hoped. There were basicly two problems

1. The OpenCV code could only run a 2 Hz
2. Flickering in the ball detection

The first problem that the code could only run at about 2 Hz because computational load of the algorithm is very heavy. The image processing is runned on the laptop which is a dualcore processor which uses in normal condition without the OpenCV about 35% of both cores, and with the OpenCV circle detection at 2 Hz over 90% of both cores. The other problem of the 2 Hz detection is that when the Turtle moves over the field there is a large vision delay which makes the Turtle move to its home-position and back the the ball, which can be explained by that that the ball position is relative old. If the Turtle sees the ball it moves its target position to the ball, but because of the low detection frequency it puts its target position back to "home" position because it does not see the ball for a long time.

The second problem is the biggest problem using the cvHoughCircles. In one sample it sees the ball and in the next sample it can not detect the ball, this is called flickering of the ball detection.

If the code is run on images taken from the camera it finds the ball or it doesn't. But when the Turtle is standing still in front of the ball, and close enough to the ball, it does not always see the ball. This so called flickering of ball detection is investigated. After a lot of testing some conclusion could be made, there are two possible reasons for the flickering. The first reason is that the images generated from the camera are not perfect identical (even when the ball is laying still and the Turtle is not moving), secondly the OpenCV code is not able to perform this real-time processing. To test our findings, a ball is put in front of the Turtle and the Turtle is set on a fixed, known position. The OpenCV code in the simulink scheme is minimized. The code now just takes a picture and does a circle detection the taken images from the camera. The results where good but gave also a not know problem. By knowing the ball postion, the cvHoughCircles code does not finds a circle at the same spot. About 50% of the time it does not find a circle at the ball postion. This may tell us that the camera images are not clear all the time. A possible reason for this is that there is an auto-adapt shutter on the camera which adapt continuously, this could explain the flickering of the ball detection. But even when the shutter is fixed the result of circle detection using cvHoughCircles where not at the ball location all the time. We suspect that the cvHoughCircles code contains some errors which should explain these strange behavior.

7 Conclusion and Recommendations

Conclusion

An arbitrary ball detection using HoughCircles method is designed, implemented and tested. In order to achieve this result basic image processing is explained where filters and edge detection methods are the most crucial parts. Furthermore the Circle Hough Transformation is explained and how it should be implemented. Testing is done in Matlab and the real-time implementation is done with OpenCV. In order to exclude incorrect ball locations, an additional check is implemented based on the shadow spot underneath the ball and the light dot on top of the ball.

This check is working really good, it can identify if a circle is really a ball. There are still some improvements needed for this code, for example a better scan for the shadow underneath the ball by taking the average green of the field into account and not a set value as has been done now. And for different distances the top and shadow positions of the ball are at other fixed pixels on the line L, as can be measured.

In general the ball detection using HoughCircle works, and should be used to find an arbitrary colored ball. This code has a heavy computing load for real-time implementation and needs to be improved to make it more efficient.

The camera on the Turtle using the Omni-vision is very good for the Turtle to find a red ball or to identify its own position on the field. But ball detection using circlehough is not recommended because the ball only has a diameter of 10 to 15 pixels depending on the distance to the Turtle. A better and more efficient way of using all the pixels of the camera is to use a front camera that gives a higher density of pixels diameter of the ball. Therefore the HoughCircle should work much better by looking at larger radii. And a smart camera where the algorithm is programmed on the camera will increase the performance of the PC.

The OpenCV code is a good step in the right direction using the cvHoughCircles for circle detection, only that it lacks in documentation which needs a lot of time to find out where all the undocumented parameters stand for. The minimum and maximum radius parameter have to be used and don't work as they should. RoboCup team (Mostly Harmles) had the same problems with the OpenCV code. We can conclude that the cvHoughCircles works but not as robust as it should be.

Recommendations

The circle detection is done on the complete image, but to make it more efficient only a small region of the image should be selected for testing on circles. This smaller region (the region of interrest) will then not effect to PC processor that hard as it does now.

There are a lot of filter techniques available, but they all need extra processor time. A perfect filter is the Bilateral filter [9]. A bilateral filter is an edge-preserving filter useful for imaging. This bilateral filter is only tested in the matlab code, and there is already needed 30 second to 1 minute to filter the image even without performing a circle detection in the image. Therefore this is not tested on the Turtle in real-time application.

Appendix A

The YUV and RGB images that the Turtle makes is taken appart. This gives a better view of each seperate channel which is shown in Figure 17.

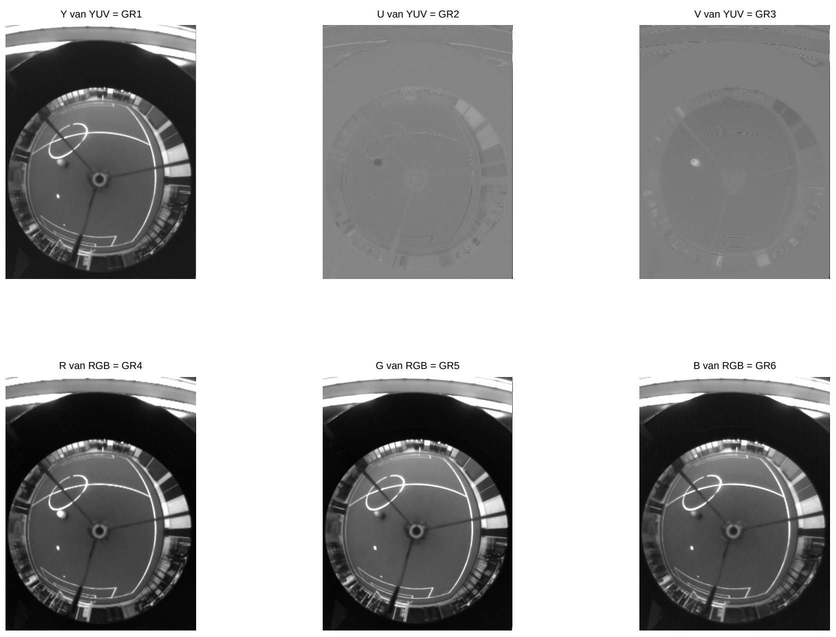


Figure 17: Seperate chanales of the YUV and RGB from the Turtle image

Appendix B

The canny edge detection is performed on the three different filters and the orginal image. The influence of the filters is now shown in Figure 18. There it shows that the blurred images loses the fieldlines and a lot of details even the ball.

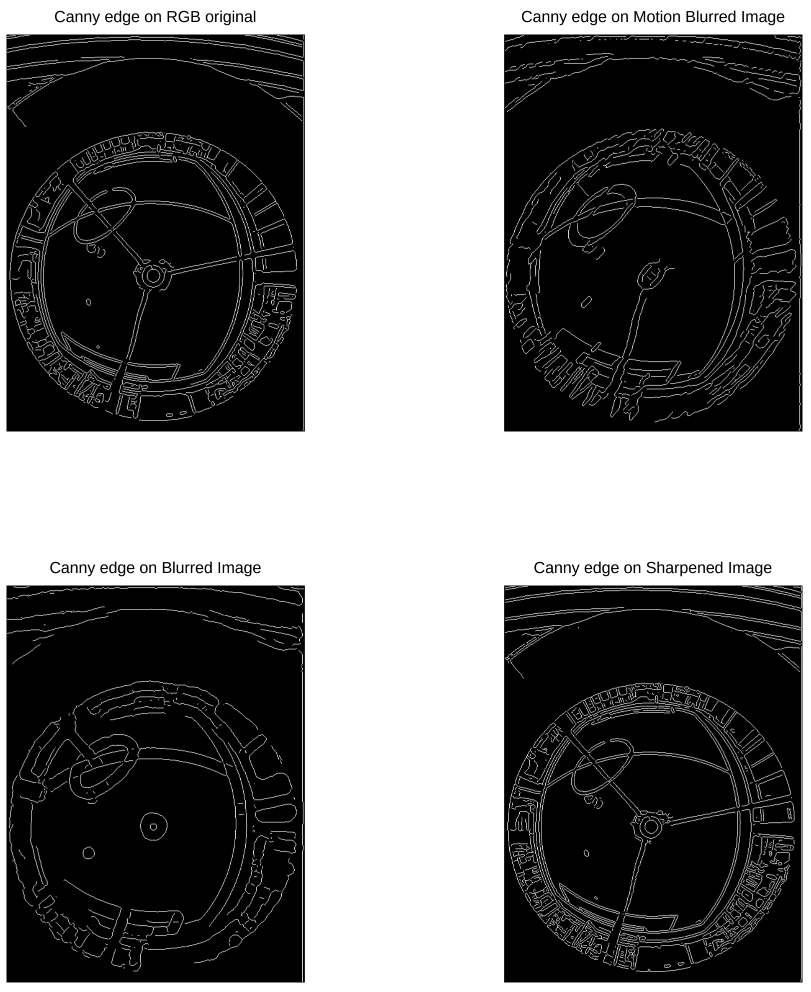

Canny edge on RGB original
Canny edge on Motion Blurred Image
Canny edge on Blurred Image
Canny edge on Sharpened Image

Figure 18: Canny edge detection on defferent filtered images

References

[1] http://www.techunited.nl.

[2] http://www.er.ams.eng.osaka-u.ac.jp/robocup-mid/index.cgi?page=Rules+and+Regulations.

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

[4] http://opencv.willowgarage.com/wiki/.

[5] http://homepages.inf.ed.ac.uk/rbf/HIPR2/wksheets.htm.

[6] http://www.mathworks.com.

[7] http://student.kuleuven.be/ m0216922/CG/filtering.html.

[8] http://www.comp.leeds.ac.uk/vision/opencv/opencvrefcv.html.

[9] http://scien.stanford.edu/class/psych221/projects/06/imagescaling/bilati.html.

[10] T.J. Atherton and D.J. Kerbyson. Size invariant circle detection. Image and Vision Computing, 17:pp. 795–803, 1999.

[11] V.A. Ayala–Ramirez, C.H. Garcia–Capulin, A. Perez-Garcia, and R.E. Sanchez-Yanez. Circle detection on images using genetic algorithms. Pattern Recognition Letters, 27:pp. 652–657, 2006.

[12] G. de Haan. Digital Video Post Processing. CIP-data Koninklijke Bibliotheek, The Hague, The Netherlands, 2006.

[13] I. Frosio and N.A. Borghese. Real-time accurate circle fitting with occlusions. Pattern Recognition, 41:pp. 1041–1055, 2008.

[14] Rafael C. Gonzalez. Digital image processing. Upper Saddle River : Pearson/Prentice Hall, 2008.

[15] T.D. Orazio, C. Guaragnella, M. Leo, and A. Distante. A new algorithm for ball recognition using circle hough transformation and neural classifier. Pattern Recognition, 37:pp. 393–408, 2004.

[16] T. Xiao-Feng, L. Han-Qing, and L. Qing-Shan. An effective and fast soccer ball detection and tracking method. National Laboratory of Pattern Recognition, 2004.

[17] H.K. Yuen, J. Princen, J. Illingworth, and J. Kittler. Comparative study of hough transformation methods for circle finding. Butterworth and Co, 8(1):pp. 71–73, februaty 1990.