Vision controlled foosball

Citation for published version (APA):

Document status and date:
Published: 01/01/2007

Document Version:
Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:
• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher’s website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:
www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:
openaccess@tue.nl
providing details and we will investigate your claim.
Vision Controlled Foosball:  
Getting started with vision

A.J.P. v. Engelen 0495318  
C.J. Zandsteeg 0535800  
DCT 2007.118

Coaches:  
Ir. J.J.T.H. De Best  
Ing. P. Brier

Supervisor:  
Prof. Dr. Ir. M. Steinbuch

Technische Universiteit Eindhoven  
Department Mechanical Engineering  
Dynamics and Control Technology Group

Eindhoven, October, 2007
# 1. TABLE OF CONTENTS

1. Introduction .................................................................................................................. 4

1. Hardware .......................................................................................................................... 5
   1.1. List of demands ........................................................................................................ 5
   1.2. Sensor ..................................................................................................................... 5
   1.3. Interface .................................................................................................................. 6
   1.4. The Camera ........................................................................................................... 7
   1.5. Objective ................................................................................................................ 8

2. Colour Spaces .................................................................................................................. 10
   2.1. Bayer pattern ....................................................................................................... 10
   2.2. RGB ..................................................................................................................... 10
   2.3. YUV ..................................................................................................................... 11
   2.4. HSV ..................................................................................................................... 13

3. Software Installation ...................................................................................................... 14
   3.1. Common issue ..................................................................................................... 14
   3.2. Prosilica Viewer Features .................................................................................. 14
   3.3. Capturing Frames ............................................................................................... 16

4. Testing the camera ......................................................................................................... 17
   4.1. Frame rates ......................................................................................................... 17
   4.2. Timestamp jitter ................................................................................................... 20
   4.3. Latency ............................................................................................................... 21

5. Lighting ........................................................................................................................ 23
   5.1. LED ..................................................................................................................... 23
   5.2. Halogen ............................................................................................................... 23
   5.3. Fluorescent lamps ............................................................................................... 23

6. Camera frame ................................................................................................................. 25

7. Image Processing .......................................................................................................... 26
   7.1. Thresholding ....................................................................................................... 26
      7.1.1 Choice of the threshold ................................................................................. 26
   7.2. Edge Detection .................................................................................................... 27
      7.2.1 Sobel ............................................................................................................. 27
      7.2.2 Canny ............................................................................................................ 28
      7.2.3 Laplace ......................................................................................................... 28
   7.3. Morphology ......................................................................................................... 30
      7.3.1 Erosion .......................................................................................................... 30
      7.3.2 Dilatation ...................................................................................................... 30
      7.3.3 Opening ........................................................................................................... 31
One of the new topics in Control Industry is Machine Vision. Machine Vision is a control technique wherein a camera system replaces sensors. Machine vision has several advantages above the use of standard sensors. For example the camera can be mounted without contacting the frame and thus cables or sensors connected to the frame, which can cause hysteresis, are not necessary anymore.

To get familiar with Machine Vision the idea rose to automate a Foosball (or table soccer) game. This is a really interesting topic because the Foosball game contains several properties which also occur in industry, such as object recognition of fast moving objects (the ball and the players). To be able to track these fast objects the vision system must run at high frame rates.

To be able to process data with high frame rates, fast computers are necessary and / or efficient algorithms have to be used. Because most computer systems do not have the arithmetic power to fulfill this demand, efficient techniques have to be used.

This leads to the primary goal of this project, to develop a fast ball recognition algorithm for the automated foosball table which is able to track the ball at a minimum of 100 fps.

To be able to achieve this goal it is first necessary to select a camera with objective. This procedure will be discussed in chapter 1 followed by used and useful colour spaces in chapter 2. After selecting a camera tests will be performed to determine latency, the influence of certain variables with respect to the frame rate and jitter on the timestamp. These tests are shown in chapter 3.

Because the vision system will be running at high frame rates with low exposure times a good light source has to be chosen to be able to capture with at least 100 fps. An overview of several light sources will be given in chapter 5. In chapter 6 a frame will be shown wherein the camera and lighting can be mounted.

To extract the ball out of the image several image processing techniques are available. These techniques are dealt in chapter 7. The application of chosen techniques to a ball recognition algorithm is explained in chapter 8.

An attempt to specify the position of the camera relative to the table is done in chapter 9, followed by an overview of problems which occurred in Chapter 10.
1. HARDWARE

As written in the introduction the game of foosball is very fast game. To be able to see the ball a camera has to be chosen. To keep track of the ball in this fast changing environment the camera should also be 'fast'. This implies a high frame rate and low shutter time. Also a free to choose region of interest (ROI) is desired to reduce redundant data transfer. This raises other questions like what interface with the computer should be used and the positioning of the camera with respect to the right lens to choose? These and other questions will be answered in this section.

1.1. LIST OF DEMANDS

In this stage of the project it is hard to get a well based list of quantitative demands. Because of the lack of own measurement data and experiments no hard boundaries can be set on for example the desired frame rate or the resolution of the frames. There is however some information available of a similar project in Germany named GIRO [Weigel,2005]. Weigel and Nebel use in their setup 50 frames per second at a resolution of 384x288 pixels in YUV format (see section 2 for more details on pixel formats). Qualitatively these figures should be 'better' in this project. A colour camera is preferred to be able to make use of colour segmentation if necessary.

1.2. SENSOR

There are basically two types of digital sensors to choose from: the CCD- (charge coupled device) and CMOS (complementary metal oxide semiconductor) sensor. The main difference between CMOS and CCD is the way individual pixels are read out. The CCD sends the charges from the pixels sequentially to a converter, which converts it to a voltage. This is then buffered and send to the output of the chip. A CMOS converts the charge to a voltage locally at each pixel [Litwiller,2001].

The main advantage of CCD over CMOS is its low noise level. This is mainly due to the on-chip circuits in a CMOS sensor, which influences the electric charge produced by each pixel, and by the common used amplifiers outside the chip on a CCD sensor, which can be larger and thus designed more precise for low noise.

These on-chip circuits at every pixel open however a wide range of possibilities for the CMOS sensors, such as timing logic, exposure control, analog-to-digital conversion, shutting, white balance, gain adjustment, and initial image processing algorithms [Turchetta,2004]. This improves the speed of the processing, but reduces the fill factor of the chip. The fill factor of a chip is the relative part of the chip which is actually covered with pixels.

Another issue is the difference in offset of the amplifiers between individual pixels in CMOS sensors. This means that in very low light conditions the pixels produce different signals, which is also known as "dark current". When there is enough light this effect is hardly visible.

The (dis)advantages described above are marginal when it comes to the foosball project. In the setup light can be added so the offset and light sensitivity problem can be tackled. The manufacturer of complete cameras is taking care of the speed advantages of the CMOS sensor. This leaves us with another important difference of the sensors; its price.

The CCD chips require a specific manufacturing method, whereas CMOS chips can be made with the same production process as mass produced memory circuits and microprocessors.
This makes the chip cheaper to produce, but the main price advantage comes from the fact that most camera functions are on the chip and no extensive printed circuit board is needed to implement these features.

All together the CMOS chip is more appropriate for the goal of this project.

1.3. INTERFACE

The next choice to make is the one of the interface; the way to transport data from the camera to the computer. To control the handlebars of the foosball table with discrete time controllers used in a computer environment, the frames have to reach the computer at certain intervals. This requires precise timing of the data to be transmitted in order to guarantee the presence of the right frame at the right time. On top of that, the data rate should be as high as possible since this can become a bottleneck to the maximum frame rate which can be achieved. Also other aspects such as ease of use and costs are important.

The three main interfaces for industrial high speed cameras are Firewire (IEEE 1394a/b), GigE Vision (IEEE 802.3 protocol) and CameraLink. The USB interface is not used much because it lacks the guarantee of timing discussed earlier unlike the three previous ones, where the timing is no problem.

Figure 1.1 shows the maximum transfer rates of different interfaces. This graph shows that CameraLink is by far the fastest technology (notice the log scale) followed by the GigE Vision standard and behind that Firewire.

![Figure 1.1: Maximum Transfer Rate Per Interface](ni.com)

The CameraLink has however some major disadvantages over GigE Vision and Firewire. The CameraLink has no industrial standard which makes the different (relative expensive) cables from different manufacturers unsuitable for other cameras. Furthermore an expensive separate
'framegraber' is needed to get the data on the computer [Prosilica, 2006]. This and the ease of use of the other two rules the CamerLink option out.

The main advantage of GigE Vision over Firewire is first the shown double bandwidth in figure 1.1. Another small advantage is the maximum length of the used cables: GigE Vision can use cables up to 100 m whereas Firewire cables are limited to 5 m [NI.com, 2006]. So the computer doesn't need to be directly next to the foosball table since already a few meters are used to guide the cable from the camera nicely to the outside. Especially the higher bandwidth is a major advantage in this project as shown in this simple example:

When RGB24 (pixel format) is used at a 640x480 resolution the frame rate is limited to

$$\frac{560 \times 10^6 \text{bits/second}}{640 \times 480 \text{pixels} \times 24 \text{bits/pixel}} \approx 130 \text{ frames per second}$$

in the GigE Vision case and to 65 frames per second in the Firewire case.

A disadvantage is that the CPU is needed to get the information from the network interface card (NIC) into the memory of the computer, while Firewire can directly send the information to the memory via the DMA controller. By choosing the right NIC (for example one with the Intel Pro 1000 chipset [NI.com, 2006]) this CPU load can be reduced.

Altogether the GigE Vision interface is preferred above the Firewire one.

THE CAMERA

Now that the type of sensor and interface is determined and there is an indication concerning the other specifications, the commercial camera is to be selected. In the department of Mechanical Engineering on Eindhoven University of Technology other projects made or make satisfactory use of Prosilica cameras. Also these cameras appeared to have a good SDK (Software Development Kit).

FIGURE1.2: PROSILICA GC-640C CAMERA [©PROSILICA, 2007]

Prosilica offers three colour cameras with GigE Vision and a CMOS chip: GE640c, GC640c and GC750c. The GC750c has a maximum frame rate of 60 fps (frames per second) at full resolution of 752x480 whereas the GE640c and GC640c have both 197 fps at full resolutions of and 659x493 pixels. This higher frame rate of the Gx640c is more important than the slightly higher resolution of the GC750c. The GC640c, shown in figure 1.2, is the compact version of the GE640c. This, according to Prosilica, shows in a smaller camera and a smaller system memory size inside the camera, but also a lower price. Therefore they suggest the GC640c above the GE640c for almost all situations, since the amount of system memory has usually no effect on the performance of the camera. See appendix 1 for details on the specifications of the camera.
1.1. OBJECTIVE

A camera is of no use without an objective (or lens). The objective should focus only the playing field on the sensor in order to make use of all pixels and no redundant data is recorded. Furthermore the objective should let as much light through as possible to minimize the amount of added light needed, because of the short exposures at high frame rates in this case. This means a large aperture size (the diameter of the ‘hole’ where the light can pass through), which maximum is given by the focal length divided by a certain number, is preferred. This so called focal ratio (or f-number) is specific for each objective and through an objective with a lower focal ratio more light passes through.

The focal length of an objective determines the working distance of the setup and thus the height of the camera above the playing field of the foosball table. The focal length may be calculated using this formula [Navitar, 2007]:

\[ FL = \frac{\text{sensor size} \times \text{working distance}}{\text{object size}} \]  \hspace{1cm} (1.1)

Formula 1.1 is derived from formula 1.2 [PolyTechnisch Zakboek, 2004]

\[ M = \frac{\text{sensor size}}{\text{object size}} = \frac{s_2}{s_1} = \frac{f}{f-s_1} \]  \hspace{1cm} (1.2)

\[ f = \frac{\text{sensor size} \times \text{working distance}}{\text{object size} - \text{sensor size}} \]  \hspace{1cm} (1.3)

The meaning of all symbols are represented in figure 1.3. Rewriting formula 1.2 results in formula 1.3. Assumed is that the object size is much larger than the sensor size and the last one thus may be neglected which is equal to formula 1.1.

![FIGURE1.3: GEOMETRIC OVERVIEW OF A LENS](image)

There are two unknowns in this equation; the focal length (FL) and the working distance. The last one is determined by the height of the camera which has to be chosen. To give enough space to the players at the table and have a practical height to reach the camera without a ladder this working distance is set to 1 m above the field. The focal length then becomes 5.7399 mm, which is close to commercially available 6mm. This brings the final working distance to 1045.3mm. The used objective is the Fujinon DF6HA-1, a 6 mm f=1.2 lens shown in figure 1.4. This lens is chosen because of its small f-number in comparison with other lenses.
FIGURE 1.4: FUJINON DF6HA-1 [©FUJINON, 2007]

※A photograph is a similar model.
2. COLOUR SPACES

In order to represent an image from a digital colour camera, the voltages of the sensors pixels need to be converted to useful information, such as standard colour spaces. Colour spaces are ways how colours of an image are represented by numbers. In the following section the source of digital images and a number of colour spaces are discussed.

2.1. BAYER PATTERN

The Bayer pattern is the basis or source for most digital images and is named after its inventor Dr. Bryce E. Bayer. His idea was proposed in 1979 and is today mostly used in single chip digital imaging devices and accordingly in the Prosilica GC640c. The Bayer pattern is an array of sensor elements on the chip which respond to red, green or blue. By placing them the way presented in figure 2.1 the number of green pixels is twice the number of red and blue ones. This is done because the human eyes response on luminance (green) is better than on the two chrominances (red and blue) [Bayer,1979]. The number behind Bayer in names for the pattern depict the number of bits used to represent each sensor element. So Bayer8 has 8 bits per pixel and can therefore have 256 different values for each individual sensor element, whereas Bayer16 can have 65536 different values per pixel.

The conversion to the colour spaces (also called demosaicking) can be done either in hardware on chip or in software on the computer. There are several algorithms like described in [Ramanath,2002] to convert the Bayer pattern into colour images, but that exceeds the scope of this project, since there is no reason for now to do it by hand on the computer.

![Figure 2.1: The Bayer Pattern](image)

2.2. RGB

The first colour space discussed is the RGB space, which is an abbreviation of 'red green blue'. The model adds the colours as they would appear when red, green and blue light is projected on a surface. In figure 2.2 squares of red, blue and green are lied upon each other to illustrate the effect of this addition of colours. An image in this space is built by three arrays (one for red, green and blue) in which each pixel has a certain number between zero and one. Zero represents no colour (black) and one represents full colour (red, green or blue). So if a pixel has value (1,0,0), it is red. (0,1,0) represents green and (0,0,1) blue. Other combinations in between are shown in figure 2.3 which is the colour space in which each pixel lies. An example is given in figure 2.4 where the three different colours are shown together with the original image.
On a computer RGB uses 8 bits per pixel per colour. This makes that there are again 256 different values for red, green, and blue. The total space contains therefore $256 \times 256 \times 256 = 16,7$ million colours. Each pixel takes thus $3 \times 8 = 24$ bits, which is often called 'True Colour'.

Some variations on the standard RGB space are BGR, RGBA and BGRA. BGR is similar to RGB, but here the red and blue components are swapped. The A in RGBA is for Alpha, which is a value for the transparency of the pixel. This does not alter the image from a camera, but can be useful when processing images at a later stage. It also adds an extra 8 bits to each pixel and a RGBA image is therefore 32 bits per pixel.

RGB has the advantage that it is easy to understand how it works and that the Prosilica camera has it as default output. The disadvantage is that it uses a lot of data (24 bits/pixel) and that a threshold (see section 7.1) is very sensitive to light conditions. Thresholding is also hard in comparison with the HSV colour space where only 2 values have to be evaluated.

1.1. YUV

The second colour space in this chapter is YUV. It uses the Y component for luminance and U and V for chrominance. This time luminance is a measure for the amount of light and is a weighted average of red, green and blue, while U is the difference of Y and R (Y-R) and V is the difference of Y and blue. The relation between YUV and RGB is shown in formula 2.1

$$
\begin{bmatrix}
  Y \\
  U \\
  V
\end{bmatrix} =
\begin{bmatrix}
  0.299 & 0.587 & 0.114 \\
  -0.14713 & -0.28886 & 0.436 \\
  0.615 & -0.51498 & -0.1001
\end{bmatrix}
\begin{bmatrix}
  R \\
  G \\
  B
\end{bmatrix}
$$

(2.1)
By choosing just one channel for the 'black and white' component, YUV can have different amount of information for Y and U/V just like the original Bayer pattern does. This can be useful with respect to the amount of data to be sent or received without (visually) losing image quality. The significance of the bits is expressed by three numbers behind YUV: YUV444, YUV422 and YUV411 for example. The first number expresses the significance of Y, the second number the significance of U and V on the even numbered image lines and the third number is for the odd numbered lines. This significance is not in terms of bits per pixel, but in terms of pixel sizes. So in YUV444 all pixels are of the same size, but in YUV422 the U and V pixels are twice the size of the Y pixels horizontally on both even and odd lines in the frame. Figure 2.6 shows the sizes of the individual pixels for the three formats which can be chosen as output on the camera. It can be seen that the resolution in horizontal direction is lower for the colour information than for the luminance in YUV422 and YUV411. This leads to less data for the total frame. For the human eye the difference between YUV444 and YUV422 is hardly visible because of the way the eye works as mentioned before. For detection methods wherein color segmentation is used a high resolution is wanted in U and V, since this is the information needed to find the object. YUV422 would basically reduce the accuracy of the recognition of the object by a factor 2.

YUV444

Y
U
V

YUV422

Y
U
V

YUV411

Y
U
V

FIGURE 2.6: PIXEL SIZES ON A SINGLE LINE IN THREE YUV FORMATS
And advantage of YUV is that shaded parts on the table only are visible in the Y channel, and therefore is YUV a robust format to use with thresholding. YUV444 is however 24 bits per pixel, so again the frame rate is limited to the capacity of Gigabit Ethernet.

2.4. HSV

The next colour space is the HSV space (hue, saturation, value). This space is more convenient with the way a human perceives a colour. Red for example stays red even if it’s not really bright or if it’s shaded. So if an object in an image is of the same colour, but one side is shaded both area’s will be in the same plane on the hue axis. Figure 2.7 shows a graphical representation of the HSV space.

The H-axis is often denoted in degrees like an angle. Just like in a circle 0° is the same as 360°. Every angle corresponds with a specific colour. It is of course also possible to run the H-axis from 0 to 1, where 0 corresponds with 0° and 1 with 360°. The S-axis gives the amount of colour present. The axis runs from zero to one, where zero means no colour (grayscale) and one full colour (no white component). The V-axis represents the value of brightness of the colour. It again runs from zero to one, where zero is black and the colour becomes brighter when increasing to 1.

The major advantage of HSV is that a colour stays on the same line inside the HSV space when its brightness changes. For example if the ball enters a shaded part of the table, which happens often in the foosball case, the Hue and Saturation of the pixels where the ball is remain the same. This makes thresholding more robust than in the RGB case. Unfortunately the camera cannot record in HSV space, so the conversion should be done on the computer. This is however a very expensive conversion in means of computations per pixel.

![FIGURE 2.7: HSV CONE](https://www.mathworks.com)
3. SOFTWARE INSTALLATION

To be able to use the Prosilica Gigabit Ethernet Camera in Windows XP some settings have to be set. This software installation procedure is described in the GC-640 manual which can be found on the website [Prosilica, 2007].

To be able to test the camera and make screenshots, Prosilica provides a program called Sample Viewer. One can find the Prosilica Sample Viewer on their website. The installation procedure can be found in the manual.

A Summary of required steps:

1. Go to: Start->Control Panel->Network Connections->Lan Connection on Intel Pro 1000 PT
2. Press right mouse button and choose properties
3. Disable all network clients and protocols except the Prosilica GigE filter and the TCP/IP protocol
4. Click once on TCP/IP
5. Choose Properties
6. Setup the right IP-address (169.254.x.y and subnet mask 255.255.0.0)
7. Press Ok
8. Configure network card by pressing the Configure button
9. Choose tab Advanced
10. Look for the option Jumbo Frames and set it to 9012.
11. Press 2 times OK

3.1. COMMON ISSUE

After installing the Gigabit Ethernet Viewer, the question is asked whether you want to install the GigE filter driver or not. The driver is necessary for some camera features, but after installing this driver it is installed in every network connection. This filter driver blocks all traffic except the traffic which is related to the camera. This can cause some problems. For example it blocks all internet traffic. Disabling this client in Control Panel->Network Connections->Right Click on the network connection -> Properties will solve this.

3.2. PROSILICA VIEWER FEATURES

When all previous steps are done properly the camera is listed in the camera window as shown in figure 3.1. Now you can select the camera and press on the  button. A window opens with the camera view.

Pressing the  button opens the camera settings. Several settings can be adapted. An overview of all settings is shown in figure 3.2 When a setting is adapted one must press enter or the  button.

By looking under the Stats section, several camera statistics can be inspected.
FIGURE 3.1: CAMERA LIST IN THE PROSILICA VIEWER

FIGURE 3.2: CAMERA CONTROL PANEL
One can adapt the camera resolution and pixel format under the section ImageFormat. Unfold the ROI (Region of Interest) section to adapt the camera resolution.

<table>
<thead>
<tr>
<th>Height</th>
<th>Number of pixels in y direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Width</td>
<td>Number of pixels in x direction</td>
</tr>
<tr>
<td>RegionX</td>
<td>Offset in x direction</td>
</tr>
<tr>
<td>RegionY</td>
<td>Offset in y direction</td>
</tr>
</tbody>
</table>

To adapt the camera’s color space, adapt the value behind PixelFormat.

The exposure (shutter speed) can be adapted under Control -> Exposure. To adapt the exposure set the ExposureMode to manual and set ExposureValue to the wanted value in microseconds. Note that the frame rate will never be higher than the inverse of the exposure time. A fixed frame rate can be set in Acquisition -> Trigger -> Fixed Rate. To run in the highest frame rate possible under the applied settings set this value to FreeRun. When the lighting conditions are not optimal the camera gain can also be changed under Gain. But note that the image quality decreases when increasing the gain on a CMOS camera.

One can save the adapted settings in the camera by unfolding the ConfigFile section, set the ConfigFile Index to the wanted position (1-4), go to ConfigFileSave and press the ConfigFileSave button. The settings can be loaded by pressing ConfigFileLoad.

3.3. CAPTURING FRAMES

Single frames can be captured by clicking with the right mouse button in the camera view window in the Prosilica Viewer software.

Frames can also be captured by developing own software with the Prosilica SDK in C++. This Software Developer Kit is also available on the website. Several samples are included to get used with all the functions.

A program called "CaptureFrames" is developed with this SDK. With this program one can capture a sequence of frames which first are written to the memory and are stored on the hard disk afterwards. This is necessary because the hard disk can’t handle such high data rates. The amount of frames which can be captured are depends on the memory size.
4. TESTING THE CAMERA

The manufacturers specifications don’t show some interesting characteristics and they will therefore be tested and discussed in this chapter by performing some synthetic tests. First the maximum frame rates for different regions of interest (ROI’s) and formats are determined, also to check whether the official specifications are met or not. Secondly the jitter on the timestamps are determined and last the latency for a frame to arrive at the pc after it’s command to get a frame is given to the camera.

4.1. FRAME RATES

In this test the maximum frame rates at different ROI’s and colour spaces are determined. To find the maximum the shutter speed is set to its lowest value and the packet size to its maximum so it won’t be the bottleneck. Furthermore the trigger mode is set to Freerunning so the camera is pushed to its maximum. See section 3 for more details on the configuration of the camera.

In figure 4.1 the number of pixels in the Y direction at a maximum number of pixels in X direction (X=659 pixels) with their corresponding frame rates and formats are shown as they were obtained from the software shipped with the camera: Prosilica GigE Viewer. The most right data points are thus the maximum frame rates at maximum resolution. The same is done for the X direction at maximum pixels in Y in figure 4.2.

FIGURE 4.1: FRAME RATES AT MAXIMUM ROI IN X DIRECTION (X=659 PIXELS)
FIGURE 4.2: FRAME RATES AT MAXIMUM ROI IN Y DIRECTION (Y=493 PIXELS)

The one thing that strikes one most is that in the Y direction with a smaller ROI the frame rate increases whereas in X direction the maximum frame rate is constant (a few measurements left aside and discussed below). This can be explained by the fact that the CMOS chip reads out the pixels row by row as stated in the datasheet [Micron, 2004]. Apparently the rows lie parallel to the X direction because the same amount of rows needs to be read out and therefore the maximum frame rate is constant for varying ROI in X direction. In Y direction a smaller ROI leads to less rows to be read out and thus a higher frame rate. This can also be seen in the equation $Frame\ Time = Number\ of\ rows \times Row\ Time$ from the chips datasheet.

The frame rates other than 198 at full resolution in figure 4.2 are due to the fact that not the chip, but Gigabit Ethernet is the bottleneck here. In table 4.1 the theoretical data rates at 659x493 are displayed for each format as well as the maximum number of frames per second (fps) which can be transferred to the computer using Gigabit Ethernet (120MB/s [NI.com, 2006]). From the table follows that even though GigE Vision is much faster than Firewire, it is still a bottleneck in high performance vision projects. Also the stated 120MB/s transfer rate by National Instruments is not achieved in the results shown in figure 4.2. This is probably due to extra overhead and operating system environment.
TABLE 4.1: THEORETICAL FRAME RATES

<table>
<thead>
<tr>
<th>Format</th>
<th>Bits per pixel</th>
<th>Data rate at 198 fps [Mbytes/s]</th>
<th>Max fps through Gigabit Ethernet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bayer8</td>
<td>8</td>
<td>62</td>
<td>403</td>
</tr>
<tr>
<td>Bayer16</td>
<td>16</td>
<td>123</td>
<td>193</td>
</tr>
<tr>
<td>RGB24</td>
<td>24</td>
<td>184</td>
<td>129</td>
</tr>
<tr>
<td>YUV444</td>
<td>24</td>
<td>184</td>
<td>129</td>
</tr>
<tr>
<td>YUV422</td>
<td>16</td>
<td>123</td>
<td>193</td>
</tr>
<tr>
<td>YUV411</td>
<td>12</td>
<td>92</td>
<td>258</td>
</tr>
</tbody>
</table>

The next thing that is to be discussed are the ‘zero’ frame rates in figure 4.2. In fact the camera crashed in these occasions. They proved to be a result of a software bug in the firmware of the camera. Prosilica acknowledged this problem and it should be fixed in the next update (Firmware 1.27).

Table 4.2 shows some additional ROIs and the corresponding frame rates in frames per second at the different formats of the camera. There is no difference in frame rates between the formats used by the camera. The bandwidth of Gigabit Ethernet is well enough under these conditions and the chip is the bottleneck here. Apparently the decoding of the Bayer pattern to RGB24 or YUVxxx in hardware is no bottleneck either, since all formats run at the same fps for a specific ROI.

TABLE 4.2: FRAME RATES AT VARIOUS ROI'S

<table>
<thead>
<tr>
<th>ROI in Y</th>
<th>ROI in X</th>
<th>RGB24</th>
<th>Bayer8</th>
<th>Bayer16</th>
<th>YUV411</th>
<th>YUV422</th>
<th>YUV444</th>
</tr>
</thead>
<tbody>
<tr>
<td>247</td>
<td>330</td>
<td>392</td>
<td>392</td>
<td>392</td>
<td>392</td>
<td>392</td>
<td>392</td>
</tr>
<tr>
<td>124</td>
<td>115</td>
<td>768</td>
<td>768</td>
<td>768</td>
<td>768</td>
<td>768</td>
<td>768</td>
</tr>
<tr>
<td>62</td>
<td>57</td>
<td>1490</td>
<td>1490</td>
<td>1490</td>
<td>1490</td>
<td>1490</td>
<td>1490</td>
</tr>
<tr>
<td>31</td>
<td>29</td>
<td>2800</td>
<td>2800</td>
<td>2800</td>
<td>2800</td>
<td>crash</td>
<td>2800</td>
</tr>
<tr>
<td>15</td>
<td>15</td>
<td>5150</td>
<td>5150</td>
<td>5150</td>
<td>5150</td>
<td>5150</td>
<td>5150</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>4850</td>
<td>4880</td>
<td>4860</td>
<td>4870</td>
<td>4860</td>
<td>4850</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4170</td>
<td>4170</td>
<td>4170</td>
<td>4170</td>
<td>4170</td>
<td>4170</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3230</td>
<td>3230</td>
<td>3230</td>
<td>3230</td>
<td>3230</td>
<td>3230</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>crash</td>
<td>2220</td>
<td>2220</td>
<td>1110</td>
<td>1110</td>
<td>crash</td>
</tr>
</tbody>
</table>

Next the frame rates with respect to the size of the packets data send over gigabit Ethernet are discussed. A packet consists of several standard bytes to initialize the data transfer and to end it as shown in figure 4.3. The GigE Vision header consists of the image number, packet number and timestamp. The actual data of the frames is denoted as x bytes and will be varied in this test. In order to let the connection between computer and camera be the bottleneck, RGB24 is used to maximize the data stream provided. Figure 4.4 shows the frame rates at various packet sizes and the percentage of the packet which is the actual information of the frames.
From figure 4.4 follows that a higher percentage of data in a packet leads to a higher frame rate. This is because relatively less redundant data such as headers and trails is send. Even though the percentage data doesn’t change that much from packet sizes of about 2000 bytes, the frame rate still increases.Apparently not only the percentage data inside a packet, but also the number of packets influences the frame rate. This is because of the extra overhead it takes to process more packets. Therefore it is best to use the largest possible packets. At a packet size of 1028 bytes the camera crashed repeatedly, which shows as a dip in figure 4.4.

4.2. **TIMESTAMP JITTER**

The next test is the jitter on the timestamp. The jitter is the irregularity in the timestamps in this case. 5000 time stamps are analyzed to investigate this noise while the camera is running at 1000 frames per second Figure 4.5 shows a histogram of the relative error with respect to 1000 frames per second in clock pulses of the internal clock of the camera. This clock runs at 66 MHz. It strikes out that only two different errors appear in the chart. This implies that in fact the camera is running at two different frame rates since the camera is working deterministic. The mean of these errors is zero, which means that on average the camera runs at the set frame rate. The cause of this behavior is still to be investigated.
4.3. LATENCY

In this test the latency is to be determined. The latency is here considered to be the time it takes before an event is seen in the pictures on the computer. To do so a test has been proposed, to use a LED which is controlled by the PC and seen by the camera. This led is attached to the RS-232 (or parallel) port. It is assumed that the LED is turned on and off infinitely fast. In practice this switching happens in microseconds and is thus much shorter than the time between frames. The LED is switched on when a certain frame has arrived in the computer (frame 100 in this situation). The software used to perform this test writes frame numbers together with the time stamp of the frame and the status of the led to a text file. By comparing the status of the led in the image and in the text file a delay of an event and the arrival of the frame registering this event can be determined. This is however a discrete time delay, since only a certain number of frames with constant time between them can be measured. In order to get a more precise latency several tests are carried out with different frame rates to vary the discreet time steps. In figure 4.6 a time path is shown to visualize the test and in table 4.3 the abbreviations are explained. The frame time is taken from the chips datasheet [Micron, 2004, Table 6], the time it takes to sent data to the computer S is a theoretical time calculated from the amount of data to be sent and the transfer rate of Gigabit Ethernet. The waiting time W is the time left after the chip ends capturing one frame and starts exposing for the second frame. The exposure time E is manually set in the configuration at 500 μS.

FIGURE 4.6: TIME SCHEME BASED ON 125 FPS AT 659*493@MONOCHROME16
TABLE 4.3: FROM FIGURE 4.6 EXPLAINED

<table>
<thead>
<tr>
<th>Abbreviations</th>
<th>Description</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>Exposure time</td>
<td>500 µs</td>
</tr>
<tr>
<td>FT</td>
<td>Frame Time; pixels are read out by chip</td>
<td>5 ms</td>
</tr>
<tr>
<td>S</td>
<td>Sending data to computer</td>
<td>±6 ms</td>
</tr>
<tr>
<td>D</td>
<td>Extra delays in computer/camera</td>
<td>4.5-5 ms</td>
</tr>
<tr>
<td>L</td>
<td>LED switched on</td>
<td>0</td>
</tr>
<tr>
<td>W</td>
<td>Camera waits for time trigger of 8 ms (125 fps)</td>
<td>±2.5 ms</td>
</tr>
</tbody>
</table>

Figure 4.7 on the next page shows four frames taken during the test at 125 fps. Due to the extreme low exposure time the images are very dark. To make the LED more visible in this report the brightness of the image is increased with 40%. In frame 102 the LED is less bright than in frame 103. This implies that the LED is switched on during the exposure time of frame 102. This is also confirmed by other tests, where a higher frame rate shows the fully lit LED at frame 103 and a lower frame rate at frame 102. Together with the assumption that the LED is switched on instantly this means that the minimal latency is twice the time used for one frame at 125 fps: $2 \times \frac{1}{125} = 16\text{ms}$ as can be seen in figure 4.6. This is the minimal latency, since the event and the exposure are at the same time. The worst case is when the event happens when the (electronic) shutter is just closed. This brings the worst case latency to $16 + FT + 11' = 23.5\text{ms}$. The delays in milliseconds only apply to this situation (full resolution, 125fps, 16bits/pixel, 500µs exposure), but give an indication on what order of magnitude the extra delays are. The exposure, frame and send time can be easily calculated so the discrete delay of 2 or 3 frames (in most cases) can also be determined, if necessary by performing the test again for the desired settings.

**FIGURE 4.7: RECORDED FRAMES DURING TEST (FULL RESOLUTION, 125FPS, 16BITS/PIXEL, 500µS EXPOSURE)**
5. LIGHTING

Another common problem in vision, especially at high frame rates, is lighting. To be able to capture at high frame rates, low exposure times are necessary. When the exposure time is decreased, also the number of photons that reach the chip in that time is decreased, under the same circumstances. Thus the image seen by the camera becomes darker and extra light has to be provided. But it is difficult to apply enough and evenly distributed light. Because in this project extra light is also necessary several light sources are tested. The results are written down in this chapter.

5.1. LED

The light emitting diode is a very efficient light source. It is supplied by a DC source so it does not have a switching frequency. Led’s are a very directional light source so it is very useful source to light small static objects. They are less suitable for large surfaces because the led’s have to be applied in a grid or array because the emitted light is very directional. Nowadays led’s are available in a very broad range of intensity.

A test is performed with a lamp containing an array of 60 bright LEDs. It did not have enough power to light a part of the table such that all objects were clearly visible at a frame rate of 100 fps. Furthermore a very large grid should be applied to light the whole playing field because of the directionality. A grid of modern high power LEDs is possibly an option, but this is very expensive and thus not tested.

5.2. HALOGEN

The halogen light source is used for high power lighting. Halogen lights can be purchased easily with powers until at least 500 W. It is not a very efficient light source, 95% of all energy is lost by heating the surrounding. Halogen light is suitable for high speed vision, because it is doesn’t free.

For the Table Soccer Case a test with halogen light is performed. The test is done with a 500 W lamp which is commonly used on construction sites. The amount of light was enough but a clear spot occurred on the field. It did not give a nice distributed light. The amount of light and generated heat is also not very comfortable for players.

5.3. FLUORESCENT LAMPS

The ordinary fluorescent lamp is a light source which can be found in almost every industrial building because of its high efficiency and low price. It is available in different color temperatures.

Old fluorescent lamps with an ordinary starting device will flicker at twice the frequency of the power net, because power being delivered to the lamp drops to zero twice per cycle. In Europe power is supplied with 230 V at 50 Hz with results in a flicker frequency of 100 Hz. This makes the fluorescent lamp with ordinary starter not very suitable for vision applications. When the frame rate with full exposure is not a multiple of this frequency different magnitudes of light flash exposures occur in different frames and thus flicker occurs. When the exposure is a multiple of this frequency the brightness of all frames is equal but one can’t say anything about the magnitude of the brightness since it depends on the moment the camera is switched on.
In more expensive fluorescent lamps the ordinary starting device is replaced by an electronic one to increase efficiency. Now the flicker frequency becomes independent of the electricity grid and the switching frequency is in the order of several kHz. This is high enough for most high speed vision applications.

A test is performed with two high frequent 58 [W] fluorescent lamps. The lighting is improved significantly. Now capturing up to 100 [Hz] is possible because to able to capture above 100 [Hz] more light is necessary, especially in the corners of the field of view due to vignetting in the lens. The difference between normal fluorescent lighting in the room and ‘fast’ fluorescent lamps is shown in figure 5.1

FIGURE 5.1: DIFFERENCE BETWEEN SURROUND LIGHTING AND MOUNTED FAST FLUORESCENT LAMPS
6. CAMERA FRAME

To have a fixed position of the camera with respect to the foosball table a rigid frame is needed. The frame can be attached using already existing holes in the table to keep its original state as much as possible at this stage of the project. The frame should not be in the line of sight of the players in any way and obey the working distance of 1045.3 mm determined by the chosen objective in section 1.5. Also the ability to mount additional lighting on the frame is desired.

The final result is a simple frame of square steel profiles, which is sectional attached to the table at four existing holes using long threaded ends. By using 40mm thick profiles the frame becomes very stiff in all directions. In figure 6.1 the table with frame and lighting is shown.

FIGURE 6.1: PICTURES OF THE FOOSBAL TABLE WITH FRAME AND LIGHTING
7. IMAGE PROCESSING

The main goal of this project is to be able to detect the ball. To be able to do so, the ball must be extracted or recognized in the image. In this chapter some image processing algorithms are explained which possibly can help to detect the ball or to remove other objects except the ball out of the image.

7.1. THRESHOLDING

Thresholding is the most basic operation in image processing. Thresholding is used to extract an object in an image when the object has different colour or intensity than the surrounding. It is used to remove irrelevant pixels in an image. Every pixel is evaluated and set to true or false. The result is a binary image wherein white the pixels belong to the relevant set. After thresholding the object becomes the only object in the binary image in the ideal case and its position can easily be calculated.

Thresholding can both be done for grayscale and color images. The difference is that grayscale images only have a 1-dimensional dataset to evaluate. Color images are 3-dimensional. To threshold a color image each color-dimension must be evaluated separately. Thus thresholding a color image takes several times more CPU cycles than the same image in grayscale.

The required computer time depends on the size and color space of the image. One of the disadvantages is that every pixel should be evaluated and thus takes much computation effort for large images. There are certain color spaces, for instance HSV, which only have to be evaluated 2-dimensional.

7.1.1 CHOICE OF THE THRESHOLD

One of the difficulties in thresholding is the choice of the threshold. The simplest manner is to take a fixed threshold. This method is not very robust, but works when the surrounding does not change in time. Another solution is to take the mean of the image. This only works when the object pixels are much brighter than the average.

When the surrounding does change in time, the histogram of the image can be used. A histogram gives an overview how many pixels belong to a certain intensity value for all values. This is very useful when the object contains much more pixels than the background or when the background values and object values are separately visible. In that case a range around a local or global peak in the histogram can be used to set the threshold value. In case of a color image the histogram contains the sum of all three dimensions in one histogram or separate histograms for each color.

In theory the histogram method works very well, but in practice when images are less uniform or contain noise it becomes difficult to find a peak in the histogram and thus the threshold values are hard to distinguish.
7.2. EDGE DETECTION

Edge detection can be very helpful when one wants to make use of geometric properties of the object to determine its position. Edge detection reduces the data in the image to edges only. It can only be applied to every colour space dimension in an image separately. Thus for 3-dimensional colour images, each colour has to be processed separately. There are several types of edge detection which can basically be subdivided in 2 categories.

The first category contains edge detection algorithms based on the gradient. When an edge occurs there is a certain slope between the intensity of the pixels. A gradient based edge detection algorithm looks for minima and maxima in the first derivative in an image.

The second category is based on the Laplacian of an image. It detects an edge by looking at zero crossings in the second derivative. The difference of the two methods can be seen in figure 7.1 [Green, 2002]

![Figure 7.1: Difference between gradient and Laplacian method [Green]]

7.2.1 SOBEL

The ‘Sobel’ edge detection is based on the gradient method. It uses two convolution masks to approximate the first derivative in two directions, one for the x-direction and one for the y-direction. Convolution masks are matrices and the values inside the matrix represent the weighting of the pixels in an image processing algorithm. With different weightings, different image processing techniques can be done. The convolution masks of the Sobel algorithm are shown in figure 7.2.

![Figure 7.2: Convolution masks for Sobel edge detection Gx (left) and Gy (right)]

The mask slides over the image and each pixel is evaluated by centering the convolution mask at that pixel. The sum of all elements of the mask and the value of the pixels gives, in this case, a
value for the gradient in one direction, \( G_x \) or \( G_y \). The total magnitude or edge strength of the gradient can be calculated by formula 7.1 or approximated by formula 7.2. This magnitude can be thresholded afterwards to remove weak edges with a magnitude smaller than a certain value.

\[
G = \sqrt{G_x^2 + G_y^2} \quad (7.1)
\]

\[
|G| = |G_x| + |G_y| \quad (7.2)
\]

The Sobel algorithm is also available in Matlab in the `edge` function in the Image Processing Toolbox. Type `BW = edge(image, 'sobel')` to apply Sobel edge detection on a loaded image in Matlab. [Green, 2002]

### 7.2.2 Canny

The Canny edge detection method uses an advanced Sobel algorithm but it is only one step in a series of steps. It gives a better result than the Sobel algorithm but it is also more expensive in computational means. The complete algorithm is explained in [Canny, 1986]

In the first step the image is filtered with a Gaussian filter to lower the influence of noise. When this filter is applied the Sobel edge detection algorithm is executed. Then the edge direction is calculated with formula 7.3. But when \( G_x \) is zero a singularity in the solution occurs. In that case the edge direction has to be equal to 90 or zero (parallel to the x-axis) degrees. It is only zero degrees if \( G_y \) is equal to zero, otherwise the edge direction is equal to 90 degrees. Because when \( G_x \) is zero an edge in y direction can be present or no edge is present.

\[
\theta = \tan^{-1} \frac{G_y}{G_x} \quad (7.3)
\]

The outcome of this formula should be rounded to the nearest possible value. Because an image is a grid of pixels not every edge direction can be represented in the image.

In the next step non-maximum suppression is applied. Non-maximum suppression suppresses pixels which are not greater than its two neighbours perpendicular to the edge direction. This will thin the edges in the output image.

Now the edges can be thresholded to remove weak edges. Two thresholds are used instead of one. A pixel in the image that has a value greater than the large threshold will be contained immediately. Pixels which neighbour to such a pixel are only contained if the intensity is larger than the smallest threshold.

The Canny algorithm is also available in Matlab in the `edge` function in the Image Processing Toolbox. Type `BW = edge(image, 'canny')` to apply Canny edge detection on a loaded image in Matlab. [Green, 2002]

### 7.2.3 Laplace

The Laplacian edge detection method uses a 5x5 convolution mask to approximate the second derivative. This convolution mask is show in figure 7.3. The Laplace Algorithm uses this mask for both x- and y-direction. A disadvantage of this method is its noise sensitivity. Because it doesn’t take the magnitude of the edge into account weak edges are also visible as zero crossings. This can be improved by a local variance measurement to make sure that a real intensity change takes place.
The Laplace algorithm is also available in Matlab in the `edge` function in the Image Processing Toolbox. Type `BW=edge(image,'log')` to apply Laplacian edge detection on a loaded image in Matlab. [Green, 2002]
7.3. MORPHOLOGY

In computer vision several types of basic image processing which are commonly used are called morphology. Morphology is mostly applied to binary images, which are images that only contain black (0) or white (1). The most important morphologic operations are dealt below. The large disadvantage of these morphologic operations is the relatively large computer time per frame.

7.3.1 EROSION:

Erosion is the first basic morphologic operation. The effect of the operation is to "wash" away the boundaries of an object in an image. In case of a binary image, objects containing white pixels will shrink in size, and holes within these objects will become larger. An example of erosion is shown in figure 7.4.

![TU/e](image1)

![TU/e](image2)

**FIGURE 7.4: EROSION WITH A STRUCTURING ELEMENT ONES(3), BEFORE (LEFT) AND AFTER (RIGHT)**

To apply erosion two objects are necessary. The first one is the image which has to be eroded. The second one is a small grid which is known as a structuring element. The size and content of the structuring element determines the final result of erosion. Every pixel is evaluated if it and its surrounding pixels meets the centre of the structuring element. When the pixel does match, the pixel is retained. Otherwise it will be erased. To apply erosion \( n \) times in Matlab use the command `Image2=bwmorph(Image,'erode',n)`. It uses a structuring element which is a 3x3 matrix filled with ones. [Fischer,2003]

7.3.2 DILATATION

Dilatation is the opposite of the erosion operation. The effect of the operation is to let boundaries of a detected object in an image grow. In case of a binary image, objects containing white pixels will grow in size and holes within these objects will become smaller. An example of dilatation is shown in figure 7.5.
FIGURE 7.5: DILATATION WITH A STRUCTURING ELEMENT \textit{ONES(3)}, BEFORE (LEFT) AND AFTER (RIGHT)

Just as the erosion operation, dilatation also needs two objects. Now when a pixel matches the center of the structuring element the surrounding pixels are set to the same values as the values for the pixels in the structuring element. To apply dilatation \( n \) times in Matlab use the command \texttt{Image2=bwmorph(Image,'dilate',n)}. The structuring element is also a 3x3 matrix filled with ones. [Fischer,2003]

7.3.3 OPENING

The opening operation is derived from the operations of erosion and dilatation. Opening can be used to remove foreground pixels that don't have a similar shape as the structuring element or that cannot contain the structuring element. Opening first applies erosion which is followed by dilation. The same structuring element is used for both operations. The effect looks much like erosion, but it is less “destructive” in general. The overall effect of opening is stipulated by the structuring element. To apply opening with a 3x3 structuring element filled with ones \( n \) times in Matlab, use the command \texttt{Image2=bwmorph(Image,'open',n)}. An example of opening is shown in figure 7.6. The white 2x2 pixel blocks disappear after one opening operation. [Fischer,2003]

FIGURE 7.6: OPENING WITH A STRUCTURING ELEMENT \textit{ONES(3)}, BEFORE (LEFT) AND AFTER (RIGHT).
Closing is the dual of opening. Closing looks much like dilatation, because it lets the boundaries of foreground pixels in an image grow. The main difference is that it preserves more of the original boundary shape. The effect of closing is again stipulated by the structuring element. The effect of closing is to preserve background regions that have a similar shape to this structuring element, or that can completely contain the structuring element. To apply opening with a 3x3 structuring element filled with ones \( n \) times in Matlab, use the command 

\[
\text{Image2} = \text{bwmorph(Image, 'close', } n)\]

An example of closing is shown in figure 7.7. The 2x2 black block inside the white pixel block and a small gap in the 'e' is filled after one closing operation, but gaps which are larger than the structuring element are retained. [Fischer, 2003]

![Figure 7.7: Closing with a structuring element ONES(3), before (left) and after (right).](image)

7.3.5 Bottom Hat

Bottom Hat is based on the closing operation (dilatation followed by erosion). When Bottom Hat is applied, first the closing of the image is calculated and then subtracted from the original image.

To apply Bottom Hat with a 3x3 structuring element filled with ones \( n \) times in Matlab, use the command 

\[
\text{Image2} = \text{bwmorph(Image, 'bothat', } n)\]

An example of Bottom Hat is shown in figure 7.8.

Bottom Hat can for example be used to find small openings in an image. In that case the opening is first closed by dilatation and secondly not opened again by erosion, thus a difference between the original image and the Bottom Hat result occurs. This difference shows the openings that are smaller than the structuring element.
7.3.6 TOP HAT

Top Hat is based on the opening of an image (erosion followed by dilatation). Applying Top Hat results in an image which is the old image minus the opening of the old image.

To apply Top Hat with a 3x3 structuring element filled with ones \( n \) times in Matlab, use the command `Image2=bwmorph(Image,'tophat',n)`. An example of TopHat is shown in figure 7.9.

Top Hat with a small structuring element can for example be used to find small objects in an image. In that case the object is first removed by erosion and not restored again by dilatation, thus a difference between the original image and the Top Hat result occurs. This difference shows the small objects and some edges in the TU/e logo.

![Figure 7.8: Bottom Hat with a structuring element \( \text{ONES}(3) \), before (left) and after (right).](image1)

![Figure 7.9: Top Hat with a structuring element \( \text{ONES}(3) \), before (left) and after (right).](image2)
7.4. HOUGH TRANSFORM

It is also possible to extract objects by its shape. This can be done by a Hough Transform. The Standard Hough Transform (SHT) is able to detect lines, but there are other transforms to detect other objects, for example circles with the Circular Hough Transform (CHT). With the Hough Circle detection the ball, which is present as a circle in the image, can be extracted. A Hough transform has to be applied at binary images, mostly the result of edge detection.

7.4.1 STANDARD HOUGH TRANSFORM

The Standard Hough Transform is developed by D. H. Ballard. It is explained in [Ballard, 1987]. The working principle of the SHT is shown below.

Every line can be written in the same format as function 7.4.

\[ x \cos \theta + y \sin \theta = \rho \]  

(7.4)

The (x,y) coordinates are known constants in the specified image. \( \theta \) and \( \rho \) are the unknown variables. Wherein \( \theta \) is the angle which the normal to the line makes with the x-axis and where \( \rho \) is the length of this normal from the origin. \( \theta \) is bounded by \([0, 2\pi]\) and \( \rho \) is bounded by the diagonal of the image.

![Figure 7.10: Coordinates in Hough Transform [Fisher, 1994]](image)

Every possible \((\rho, \theta)\) combination at a certain \((x,y)\) position in the image can be drawn as a sinusoid in the Hough parameter space. Multiple intersections of sinusoids in the Hough parameter space represent a line in the image with the intersection point coordinates \((\rho, \theta)\). An example can be found in figure 7.11. Note that the origin of the line lays in the lower left corner. All lines for all white pixels are plotted. The intersection points lay at \(\frac{\pi}{4}\) pi and \(\frac{7}{4}\) pi which represent the same line.
FIGURE 7.11: STANDARD HOUGH TRANSFORM (RIGHT) OF A DIAGONAL LINE STARTING IN (0,0) (LEFT)

By selecting a stepsize the Hough Parameter space becomes a finite matrix. This Matrix is called the Accumulator. The size of the Hough parameter space can be chosen by selecting $\theta$ and $\rho$ with a certain step size. The larger the step size, the faster the algorithm. [Fischer, 1994]

The SHT is available in the MATLAB® Image Processing Toolbox by using the function `hough(bw)`.

### 7.4.2 CIRCULAR HOUGH TRANSFORM

The Circular Hough Transform (CHT) is another variant to detect circles in an image. It can be used to find parameters of a circle when some points on the perimeter are known. A circle can be described by equations 7.5 and 7.6.

\[
X = a + R \cos(\theta) \quad (7.5)
\]
\[
Y = b + R \sin(\theta) \quad (7.6)
\]

wherein the coordinates of the centre of the circle are $(a, b)$, the radius is $R$ and the angle is $\theta$ which takes values in the interval $[0, 2\pi]$. When points in an image lay on the perimeter of a circle with certain parameters the CHT can be used to find the parameters $(a, b, R)$. The disadvantage of this technique is that the parameter space is 3-dimensional which makes it an expensive algorithm.

To reduce computation time an adaptation to this algorithm can be made when the radius of the circle is known. Now the unknown parameter space in equation 7.5 and 7.6 becomes 2-dimensional where $(a, b)$ are the unknowns. When a circle with a radius $R$ is present in an image, circles with the same radius which have centre points on the perimeter of the original circle, intersect in the centre of the original circle. Thus the centre point can be found with a Hough accumulator. This is illustrated in figure 7.12. [Rhody, 2005]

A CHT algorithm is not available in Matlab. It can be found in the Matlab File Exchange written by Amin Sarafrz see [Sarafrz, 2004].

35
FIGURE 7.12: ILLUSTRATION OF THE CHT WITH FIXED RADIUS [RHODY, 2005]
8. BALL RECOGNITION

One of the goals of this project is to recognise objects, the ball in this case, at high frame rates. To be able to achieve this, the ball recognition algorithm should be as simple as possible to minimize the calculation effort per frame.

One of the fastest image processing tools is the previously mentioned thresholding technique. This can be achieved when the ball has a completely different colour or intensity than the other objects on the table. For the table soccer case a red ball is used.

At first thresholding with a colour range around the ball colour was applied in RGB space. The mean position of all pixels after thresholding is used as the ball position. This method was not very robust in practice. Edges around other objects contained also red pixels which disturbed the measurement and caused a large error.

To make the algorithm more robust it is extend with a Circular Hough Transform with fixed radius. In general the Hough Transform is not applied to thresholded images, but only to edge detections. But edge detections are computationally expensive and thus not suitable for real-time processing at high frame rates on standard equipment. A thresholded image is used as a replacement.

The assumption is made that there are no larger object with similar colour in this case. A disadvantage of the use of a thresholded image instead of an edge detection is that the object is not an line but a solid. When Hough is applied in such manner also larger objects with the same colour can be seen as a ball.

An advantage is that the diameter of the ball doesn’t have to be known exactly. Thus when the ball diameter differs slightly due to lens-distortions or when the ball is kicked upwards, it still can be detected as long as the given radius in the Hough transform is smaller than the radius in the image under all circumstances.

When this is implemented an optimised for speed in MATLAB® it runs at approximately 20 frames per second on a Intel Core 2 Duo E6600 processor.

8.1.1 ENHANCEMENTS

Because thresholding followed by a Hough transform is too slow, speed improvements are necessary. Thus the number of calculations necessary should be decreased. This can be done by reducing the image size and thus the number of pixels. To achieve this, a Region of Interest is introduced.

The first frame is processed completely to detect the initial position of the ball. When this position is known a small square region around this position is extracted for the next frame. The dimensions of this region are depending on the maximum distance which the ball can travel in one frame. This is a linear combination of the frame rate and the maximum ball velocity. This dependency is described by formula 8.1.

The maximum ball velocity is determined in a test. An experienced player is asked to shoot the ball with a velocity as high as possible. This test is recorded and the number of pixels the ball travels per frame is calculated. This distance is also measured in [m] and calculated in [m/s]. The result is a maximum ball velocity of approximately 7.5 [m/s]. For the rest of this project a maximum ball velocity of 10 [m/s] is used to be sure it can always track the ball. After this
enhancement frame rates of 200 Hz can be achieved. The MATLAB code is shown in appendix 2. The application on a sequence of images can be seen in Appendix 3.

\[
ROI = \frac{\text{amount of pixels} \cdot \max \text{ velocity}}{\text{field size} \cdot \text{framerate}} \quad \text{[Pixels]} 
\]  

(8.1)

In Appendix 3 a sequence of recorded images is shown with the results of the ball recognition algorithm.

The image on the left is the image as seen through the camera. The blue line is the line of movement and the red arrow represents the direction and magnitude of the velocity vector. These are both calculated in the m-file for each frame. The blue spot represents the calculated centre of the ball.

The image on the right is the thresholded ROI, wherein the blue spot the calculated centre of the ball represents.

As one can see in the images the ball is recognized all the time with small error. When the bar occludes the ball it is also recognized, because of the speed prediction. Also other objects don't disturb the measurement such as a human hand or a ball with different colour.

Other sequences are also evaluated wherein the ball bounces against the wall and the algorithm still detects its position for every frame.
9. POSITION

The position of the camera relative to the table should also be known to be able to translate the position of the ball in an image to the position of the ball on the field. To calibrate the cameras position relative to the table a Hough line detection algorithm can be performed to locate lines in the field. The position of these lines is known on the table, thus with the Hough results the position of the camera relative to the table can be calculated. This method can be used to correct for errors in the camera position, but not for determining the real position of the bal.

The disadvantage of this method is that a small error of several pixels can occur because the lines in the field are wider than 1 pixel, for example 4 pixels. The line found with the Hough algorithm is somewhere on this line, thus can have an error of 2 pixels for example.

When the camera is mounted in the stiff frame as described in chapter 6, its movements are that small that the uncertainty of the line detection method is larger than the camera's movements. When mounted in the frame the connection of the camera to the table can be assumed to be rigid.
10. ISSUES

10.1. OCCLUSION

Because the camera is mounted on top of the table occlusion of the ball can occur. This is due to the bars with the players on it. This is a big problem because occlusion can cause discontinuities in the ball detection algorithm. One way to solve this problem is to implement zero order hold of the position below a certain confidence level in the Hough transform. This is the easiest way to deal with occlusion. This works very well in theory, the previous ball position is kept and when the ball is not occluded anymore the algorithm can easily detect it again because the ball should still be in the Region of Interest. A disadvantage of this method is that it destroys the derivative of position because velocity is zero when the position is held.

A better method is to predict the ball position with use of its velocity and direction. Now the new ball position is there where it should be expected. This method prevents zero's in the derivative of position. A disadvantage is that when the ball bounces against a player or a wall, the position is predicted in the wrong direction. But normally in these cases a ball prediction is not necessary because the ball is not occluded then

Note that occlusion can be prevented by mounting the camera below the table with a translucent field.

10.2. VIGNETTING

A common problem in vision projects is vignetting of the lens. It's the effect that images become less bright towards the edges of the image compared to a certain circle in the centre. Figure 10.1 shows this on a white sheet.

![Figure 10.1: Vignetting](image)

It is caused by the aperture size of the lens. A large aperture size shows more vignetting than a small one. Due to the dimensions of the aperture, a point which is not in the center of the lens 'sees' a smaller aperture than a point on the centerline. In figure 10.2 one can see that for aperture AA' the ray from axial point 'sees' a larger aperture than a ray from a point at angle $\alpha$, which is BB'. With aperture BB' the points from angles smaller than $\alpha$ there is no vignetting effect [Aggarwal,2001].
So a smaller aperture size decreases the effect of vignetting, but also reduces the amount of light traveling through the lens and thus more light is needed. A tradeoff has to be made between the extend of vignetting and the amount of light added.

Figure 10.3 shows the effect on the foosball table. One bar has been removed to illustrate the additional effect of vignetting and shades. At the right there is a clear line behind which the table looks much darker. This could be a problem for the ball recognition algorithm and has to be taken into account.
11. CONCLUSIONS

Entering a new topic such as machine vision raises a lot of questions. In this report several subjects were presented, such as the choice of a suitable camera and techniques to analyse images. To get more insight in this topic and get practical experience all is applied on a foosball table and a setup to do experiments has been designed and image processing techniques are discussed in order to find the ball in this game.

First some insight has been given into the several "variables" which are present by selecting a camera. The difference between CMOS and CCD sensors is the way the pixels are read out, which both have advantages and disadvantages. The main conclusion of this part is that CMOS sensors and its control circuit are cheaper than CCD sensors and if the amount of light is not a problem the CMOS sensor is preferred because of its price.

The second choice concerning the camera is the interface to the computer. When a connection has to be chosen the GigE Vision standard is preferred because of its high speed and it is a well chosen standard. But remind that this technology and thus speeds increase rapidly so maybe better standards are available in the near future.

In order to get a feeling of the abilities of the camera three experiments have been carried out. One of the test is the determination of the maximum latency between an event to happen to arrive in the memory of the computer. The latency is depending on image size, colour space frame rate. For a full resolution frame with greyscale colour space is 16-23 milliseconds or 2 or 3 frames.

The jitter on the timestamp is also investigated. The outcome of this test is that the camera not runs at the exact frame rate as set, but at 2 frame rates actually. All frames can be divided in two sequences of frames. One sequence which has a frame rate below the frame rate as set, and another sequence which has a frame rate above this value. The mean frame rate of these two sequences is the frame rate as set. The reason for this behaviour is still to be investigated.

The last synthetic experiment is to determine the maximum frame rates in different circumstances. The dependency of several factors to the frame rate is quite as expected. When the packet size drops, also the frame rate drops due to overhead. When the x-resolution increases, the frame rate decreases exponentially (except for really small resolutions). When the y-resolution increases, the frame rate is almost static. This is because the used chip in the camera reads the pixels in rows. It has no influence on the time it takes to read such a row if a row is only partially evaluated. The used colour space does influence the maximum frame rate, because GigE Vision has a limited bandwidth. RGB uses one third more data per frame then the Bayer16 setting, which causes limitations in the number of frames which can be send over GigE Vision.

The next item in the setup is the objective (or lens) for the camera. It is explained how to select the right objective. Also insight is given into the term F-number (diaphragm) and that it is related to the vignetting effect. The larger the diaphragm, the more vignetting is visible.

The last part is the lighting, which is improved such that that the highest frame rate with good image is doubled. To achieve this fluorescent lamps with high frequent electric starters are used.
Now that there is a working setup, the focus is on image processing. Several image processing techniques are discussed, but most techniques require too much arithmetic power that it is too slow even on a high performance personal computer. Nevertheless a ball recognition algorithm is developed by merging two image processing techniques: thresholding and Hough Circle Detection. An ROI has to be used to reduce the amount of data such that processing at high frame rates is possible. Then the ROI is thresholded and a Circular Hough Transform is applied afterwards. Frame rates up to 200 Hz are possible now. The algorithm is extended with a velocity prediction and it is robust for occlusions.

Since this was the first step in controlling the foosball table, a lot has to be done to complete the project. For example the exact position of the ball on the field has to be determined by calibrating the lens. Also all tests and ball recognition that have be done, were done offline. A real time connection from the camera to the algorithm has to be designed to get more results from real playing situations over a longer period of time.
12. RECOMMENDATIONS

To improve the algorithm as described in section 8 some parts still have to be investigated yet. At first the first frame causes a problem because it requires much more computer time than the other frames. To avoid this problem the ball can be brought into the game at a fixed position. When a goal is scored or a game is started the ROI can be set at this hole and thus the required computer time becomes equal for all frames.

Another issue is the used colour space. As written in section 2.5 the YUV colour space has the advantage that the threshold becomes light independent. This is not implemented yet in the algorithm and also software has to be written to grab the frames in YUV format.

To improve lighting a reflecting cover can be placed above the fluorescent lamps. Now much light is radiated to the ceiling of the room the table is placed in. The cover can reduce this loss significantly.

Also the remark has to be made that the described algorithm only is tested offline. Thus first the frames are stored on the computer and processed afterwards in Matlab. The interface to process GigE vision realtime in Matlab is not available yet.
REFERENCES


45
[Weigel, 2005] T. Weigel, "KiRo - A Table Soccer Robot Ready for the Market", *Künstliche Intelligenz*, Issue 01/05, 2005
## APPENDIX 1: GC640C SPECIFICATIONS

<table>
<thead>
<tr>
<th>Specification</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resolution</td>
<td>659x493</td>
</tr>
<tr>
<td>Sensor</td>
<td>1/2&quot; CMOS progressive scan</td>
</tr>
<tr>
<td>Pixel Size (μm)</td>
<td>9.9 x 9.9</td>
</tr>
<tr>
<td>Maximum Frame Rate (full resolution)</td>
<td>197 fps</td>
</tr>
<tr>
<td>Lens Mount</td>
<td>C-mount</td>
</tr>
<tr>
<td>Digital Interface</td>
<td>GigE Vision 1.0</td>
</tr>
<tr>
<td>Interface Type</td>
<td>IEEE 802.3 1000baseT</td>
</tr>
<tr>
<td>Exposure Range</td>
<td>20μs to 60s</td>
</tr>
<tr>
<td>Gain Range</td>
<td>0 to 16dB</td>
</tr>
<tr>
<td>Region of Interest (ROI)</td>
<td>Independent x and y control; 1 pixel resolution</td>
</tr>
<tr>
<td>Frame Rate at 100 x 100 ROI</td>
<td>935 fps</td>
</tr>
<tr>
<td>Binning</td>
<td>N/A</td>
</tr>
<tr>
<td>Imaging Modes</td>
<td>External Trigger, Fixed frame rate, Software trigger</td>
</tr>
<tr>
<td>Fixed Frame Rate Control</td>
<td>0.001 fps to maximum frame rate</td>
</tr>
<tr>
<td>External Trigger Modes</td>
<td>Rising edge, Falling edge, Any edge, Level high, Level low</td>
</tr>
<tr>
<td>External Sync Modes</td>
<td>Trigger ready, Trigger input, Exposing, Readout, Imaging, Strobe, GPO</td>
</tr>
<tr>
<td>Trigger Delay Control Range</td>
<td>0 to 60s in 1 us increments</td>
</tr>
<tr>
<td>Trigger Latency</td>
<td>6.1 μs</td>
</tr>
<tr>
<td>Trigger Jitter</td>
<td>+/-20ns</td>
</tr>
<tr>
<td>External Trigger/Sync Connection</td>
<td>12-pin Hirose</td>
</tr>
<tr>
<td>Monochrome Modes</td>
<td>Mono8</td>
</tr>
<tr>
<td>Color Modes</td>
<td>Bayer8, Bayer16, RGB24, YUV411, YUV422, YUV444, BGR24, RGBA24, BGRA24</td>
</tr>
<tr>
<td>GPIO</td>
<td>1 isolated TTL input, 1 isolated TTL output, 1 non-isolated TTL input, 1 non-isolated TTL output, RS232 I/O</td>
</tr>
<tr>
<td>Max. Power Consumption</td>
<td>2.3 W (12V)</td>
</tr>
<tr>
<td>Specification</td>
<td>Value</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>Max. Operating Temperature</td>
<td>50 °C</td>
</tr>
<tr>
<td>Housing Size (not including lens mount and connectors)</td>
<td>33x46x33 mm</td>
</tr>
<tr>
<td>Total Size Envelope (HxWxL)</td>
<td>33x46x51 mm</td>
</tr>
<tr>
<td>Weight</td>
<td>92g</td>
</tr>
<tr>
<td>Conformity</td>
<td>CE, FCC, RoHS</td>
</tr>
<tr>
<td>Digitization</td>
<td>10 bits</td>
</tr>
<tr>
<td>Spectral Sensitivity Range</td>
<td>400 - 1000 nm</td>
</tr>
</tbody>
</table>
APPENDIX 2: MATLAB CODE FOR BALLRECOGNITION

A2.1 HOW TO USE BALLRECOGNITION.M

- Load image sequence with loadimages.m and save this into a .mat file
- Make sure that this file is loaded inside ballrecognition.m
- Optionally change the threshold values
- Run

A2.2 BALL RECOGNITION M-FILE: BALLRECOGNITION.M

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Ballrecognition.m
% authors:
% A.J.P van Engelen, C.J. Zandsteeg
% Department of Mechanical Engineering
% Eindhoven University of Technology
% June 2007
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%% Matlab Environment Initialisation
clc
clear all
global ROI

%%%% load image sequence
load constantspeed.mat
load changeofdirection.mat
disp('init: ')
tic

%%%% init
%%%% load first image
image_init=squeeze(images(1,:,:,:,:));

%%%% determine size of image
[m n p]=size(image_init);
n_o_f=length(images(:,:,1,1,1)); %number of frames

%%%% RGB threshold range
R=uint8([230 255]);
G=uint8([0 140]);
B=uint8([0 30]);

%%%%Declaration of variables to improve speed
threshold_init=zeros(m,n);
vx=zeros(n_o_f-1,1);
vx=zeros(n_o_f-1,1);
vx=zeros(n_o_f-1,1);
vx=zeros(n_o_f-1,1);
vx=zeros(n_o_f-1,1);
x0detect=ROI;
y0detect=ROI;

%%%% possibility to read colors when light is ideal.
% Calculate Region of Interest
framerate=input('Framerate (fps) ?:');
ROI=6000/framerate;  % ROI is based on a maximum ball speed of 10 m/s

% Determine first ball position, analysing complete frame
toc
disp('full')
tic

% Thresholding in RGB space
for y1=1:m  % y (vertical) direction
    for x1=1:n  % x (horizontal) direction
        if image_init(y1,x1,1) > R(1)  % RGB Red Threshold
            if image_init(y1,x1,3) < B(2);  % RGB Blue Threshold
                if image_init(y1,x1,2) > G(1) & image_init(y1,x1,2) < G(2);  % RGB Green Threshold
                    threshold_init(y1,x1)=1;  % Binary matrix, 1= pixel with right color
                end
            end
        end
    end
end

% Hough circle detection
[y0,x0,Accumulator,maximum] = houghcircle(threshold_init,4,15,vx(1),vy(1));
toc

% Ball recognition with ROI
for q=2:n_o_f
disp('ROI')
tic

% clear threshold matrix
threshold=zeros(2*ROI,2*ROI);

for y1=max(1,(y0(q-1)-ROI)):min(m,(y0(q-1)+ROI))  % y (vertical) direction
    for x1=max(1,(x0(q-1)-ROI)):min(n,(x0(q-1)+ROI))  % x (horizontal) direction
        if images(q,y1,x1,1) > R(1);  % RGB Red threshold
            if images(q,y1,x1,3) < B(2);  % RGB Blue threshold
                if images(q,y1,x1,2) > G(1) & images(q,y1,x1,2) < G(2);  % RGB Green threshold
                    threshold(y1-(y0(q-1)-ROI)+1,x1-(x0(q-1)-ROI)+1)=1;  % Binary matrix, 1 = a pixel with right color
                end
            end
        end
    end
end

% Hough circle detection
[y0detect,x0detect,Accumulator,maximum] = houghcircle(threshold,8,20,vx(q-1),vy(q-1));  % apply Hough Circle Transform

% conversion local position to global position
x0(q)=x0detect+(x0(q-1)-ROI);
y0(q)=y0detect+(y0(q-1)-ROI);
% Calculate velocity (differentiating)
vx(q)=x0(q)-x0(q-1);
v_y(q)=y0(q)-y0(q-1);

% save unchanged velocity vector
vx_o(q)=vx(q);
v_y_o(q)=vy(q);

% averaging velocity over 3 samples
vx(q)=sum(vx(max(1,q-2):q))/length(vx(max(1,q-2):q));
v_y(q)=sum(vy(max(1,q-2):q))/length(vy(max(1,q-2):q));

% Visualize direction and speed, calculating line of movement
if vx(q)==0
    rc=0;
else
    rc=vy(q)/vx(q);
end
b=-(rc*x0(q))+y0(q);
x=x:n;
toc

% plot results:
figure(1); hold off; subplot(1,2,1)
imshow(squeeze(images(q,:,:,1))); % plot rgb-image
hold on; plot(x0(q),y0(q),'bo'); plot(x0(q),y0(q),'b+'); % plot marker (global position)
plot(x,rc*x+b);
quiver(x0(q),y0(q),50*vx(q),50*vy(q),',r', 'linewidth',2, 'MarkerSize', 50)
subplot(1,2,2) % plot threshold matrix
imshow(threshold)
hold on; plot(x0detect,y0detect,'bo'); plot(x0detect,y0detect,'b+') % plot marker in threshold matrix (local position)
end

A2.3 HOUGH CIRCLE DETECTION M-FILE: HOUGH CIRCLE.M

function [y0detect, x0detect, Accumulator, Value] = houghcircle(Imbinary, r, thresh, v0x, v0y)

%HOUGH CIRCLE - detects circles with specific radius in a binary image.
%   
%   Comments:
%   Function uses Standard Hough Transform to detect circles in a binary image.
%   According to the Hough Transform for circles, each pixel in image space
%   corresponds to a circle in Hough space and vise versa.
%   Upper left corner of image is the origin of coordinate system.
%   
%   Usage: [y0detect, x0detect, Accumulator] = houghcircle(Imbinary, r, thresh)
%   
%   Arguments:
%   Imbinary - a binary image. image pixels that have value equal to 1 are
%   interested pixels for HOUGH CIRCLE function.
%   r - radius of circles.
%   thresh - a threshold value that determines the minimum number of
%   pixels that belong to a circle in image space. threshold must be
% bigger than or equal to 4 (default).
% vox - velocity of previous sample in horizontal direction
% (default=0), used as a prediction < thresh.
% v0y - velocity of previous sample in vertical direction
% (default=0), used as a prediction when < thresh
%
% Returns:
% yidetect - row coordinates of detected circles.
% xidetect - column coordinates of detected circles.
% Accumulator - the accumulator array in Hough space.
% Value - number of pixels that belong to the circle
%
% Written by:
% Amin Saraqraz
% Photogrammetry & Computer Vision Devison
% Geomatics Department, Faculty of Engineering
% University of Tehran, Iran
% saraqraz@geomatics.ut.ac.ir
% May 5, 2004 - Original version
% November 24, 2004 - Modified version, faster and better performance
%
% Edited by:
% C.J. Zandsteeg
% A.J.P.v.Engelen
% Department of mechanical Engineering
% Eindhoven University of Technology, The Netherlands
% June 20, 2007 - Modified by Zandsteeg & v. Engelen

global ROI

%% Checking Arguments in
if nargin == 2
    thresh = 4;
elseif thresh < 4
    error('threshold value must be bigger or equal to 4');
end
if nargin == 3
    v0x=0;
    v0y=0;
else

%% Voting
Accumulator = zeros(size(Imbinary));
[yIndex xIndex] = find(Imbinary);
for cnt = 1:length(xIndex)
    low=xIndex(cnt)-r;
    high=xIndex(cnt)+r;
    if (low<1) low=1; end
    if (high>size(Imbinary,2)) high=size(Imbinary,2); end
    for x0 = low:high
        y01 = yIndex(cnt)-sqrt(r^2-(xIndex(cnt)-x0)^2);
        y02 = yIndex(cnt)+sqrt(r^2-(xIndex(cnt)-x0)^2);
        y01 = round(y01); y02 = round(y02);
        if y01 < size(Imbinary,1) && y01 >= 1
            Accumulator(y01,x0) = Accumulator(y01,x0)+1;
        end
        if y02 < size(Imbinary,1) && y02 >= 1
            Accumulator(y02,x0) = Accumulator(y02,x0)+1;
        end
    end
end
%% Finding global maximum in Accumulator
[I, yldetect] = max(Accumulator);
[Value, xldetect] = max(I);
yldetect = yldetect(xldetect);

%% Prediction
if Value < thresh
    disp('probably not a ball, using prediction ')
yldetect = ROI+round(v0y);
xldetect = ROI+round(v0x);
end

%2.4 LOAD IMAGE SEQUENCE M-FILE: LOADIMAGES.M

function images=loadimages(number, first)

%% Image loader
%% A.J.P. van Engelen, C.J. Zandsteeg
%% Department of Mechanical Engineering
%% Eindhoven University of Technology
%% June 2007

% Usage:
% Make sure the images are named in %04d.tif format
% for example: 0001.tif or 0973.tif and in current directory
%
% images=loadimages(100); loads 0001.tif till 0100.tif in images
%
% images=loadimages(40,150); loads 0040.tif till 0150.tif in images

if nargin==1
    first=0;
    last=number-1;
end

if nargin== 2
    last=first;
    first=number-1;
end

for i=1:last-first
    name = sprintf('%04d.tif',i+first);
    images(i,:,:,:) = uint8(imread(name,'tif'));
end
## APPENDIX 3: IMAGE SEQUENCE OF BALLRECOGNITION.M

<table>
<thead>
<tr>
<th></th>
<th><img src="image1.png" alt="Image 1" /></th>
<th><img src="image2.png" alt="Image 2" /></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><img src="image3.png" alt="Image 3" /></td>
<td><img src="image4.png" alt="Image 4" /></td>
</tr>
<tr>
<td>2</td>
<td><img src="image5.png" alt="Image 5" /></td>
<td><img src="image6.png" alt="Image 6" /></td>
</tr>
<tr>
<td>3</td>
<td><img src="image7.png" alt="Image 7" /></td>
<td><img src="image8.png" alt="Image 8" /></td>
</tr>
<tr>
<td>4</td>
<td><img src="image9.png" alt="Image 9" /></td>
<td><img src="image10.png" alt="Image 10" /></td>
</tr>
</tbody>
</table>