Unterschrift (Betreuer)

TECHNISCHE UNIVERSITÄT WIEN

VIENNA UNIVERSITY OF TECHNOLOGY

DIPLOMARBEIT

Feature-Based Real-Time Stereo Vision on a Dual Core DSP with an Object Detection Algorithm

Ausgeführt am

Institut für Rechnergestützte Automation, Arbeitsbereich für Mustererkennung und Bildverarbeitung

der Technischen Universität Wien

unter der Anleitung von

a.o. Univ.Prof. Dipl.-Ing. Dr.Robert Sablatnig

durch

Markus Bader

Lindengasse 3/12, 1070 Wien

Wien, Februar 2006

Unterschrift (Student)

Kurzfassung

Objekterkennung, die Lokalisierung der eigenen Position und das Kartographieren eines Raumes sind Problemstellungen, die in der Robotik durch den Einsatz eines Bildsensors gelöst werden können. Ein Roboter namens Tinyphoon wurde im Rahmen dieser Diplomarbeit mit einem Bilderkennungssystem ausgestattet. Es werden Algorithmen und Methoden präsentiert, die bei der Implementierung dieses Bilderkennungssystems verwendet wurden, wodurch es dem Roboter ermöglicht wird, an einem Roboterfussballtunier teilzunehmen. Das Ziel dieser Arbeit ist es, für eine gegebene Hardware optimale Prozeduren zu finden und zu entwickeln, um 3D-Informationen aus aufgenommen Bildern zu extrahieren. Diese 3D-Informationen bestehen aus Linien im dreidimensionalen Raum und den Positionsinformationen bekannter erkannter Objekte am Spielfeld wie Spielball und Tore. Die dazu verwendete Hardware basiert auf einen Dual Core DSP mit 256 Kbyte Cache und 32 MB SDRAM. Zwei Kameras, wie sie auch in kommerziellen mobilen Telefonen eingesetzt werden, sind im Abstand von 30 mm an der Front des Bilderkennungssystems montiert. Durch die kompakte Bauweise des Roboters ist die Größe des Bilderkennungssystems auf 75 mm x 75 mm beschränkt.

Die Arbeit besteht aus vier Teilen. Der erste Teil beschäftigt sich mit dem Erkennen von Kanten und Linien in Bildern. Dabei kommt ein neuer, speziell für eingebettete Systeme optimierter Linienerkennungsalgorithmus zu Einsatz, welcher auf lokalen und globalen Linienparametern basiert.

Ein Feature-Based Stereo Vision Algorithm wird im zweiten Teil der Arbeit beschrieben. Dieser verwendet als Features die im ersten Teil extrahierten Kanten und Linien, um Tiefeninformationen zu berechnen.

Der dritte Teil beschäftigt sich mit einer auf Farben basieren Bildsegmentierung, um Blobs zu detektieren. Diese Segmentierung sowie die erkannten Kanten und Linien werden im letzten Teil genutzt, um Objekte und deren Position zu ermitteln. Als Testumgebung wird ein Roboterfussballspielfeld verwendet, in dem der Spielball, die Tore, die Roboter sowie die Linien der Bodenmarkierungen erkannt werden. Sowohl die Tore als auch die Roboter werden anhand ihrer Farbe erkannt. Des Weiteren wird der Spielball mit Hilfe eines Kreiserkennungsalgorithmus vermessen. Zudem enthält die Implementierung eine einfache Rektifizierung. Die vom System erreichte Bildwiederholrate ist abhängig vom gewählten Bildauschnitt und liegt zwischen 5 Hz und 11 Hz.

Abstract

A vision sensor mounted on a robot enables the robot to solve problems such as object detection, self-localization and room-mapping with one sensor. A robot named Tinyphoon is equipped with a vision system to make it possible for the robot to play robot soccer. This thesis proposes the work steps and methods used to implement this vision system. The implementation is realized using a Dual Core DSP with 256Kbyte cache, 32MB SDRAM and two cameras directly connected to the processor. The cameras are similar to the one used in common cellular phones and mounted with a base-line length of 30mm. The mechanical size of the sensor is circular and measures 7.5cm in diameter. The subject of the author's master thesis is to select, implement and design fast algorithms for the given hardware, to extract 3D lines and the positions of known objects.
The work can be divided into four parts. The first part is the implementation of a feature detection algorithm to find edges and lines. Lines are detected by a new line detection which is based on local and global line parameters and optimized for embedded systems. The second part is a feature-based stereo vision algorithm using the edges and lines of the previous part. A color segmentation with a blob detection is the third part. Objects in the last part are localized by using blob information and measured by lines and edges. Therefore, a circle detection is implemented. The detection of rectangular shapes is not part of this thesis but rectangular objects can be detected by their color. The set of the extracted objects is known and limited. A robot soccer environment are to be used as a test field where the game ball, the players, the goals and the landmarks on the playing field are to be detected. The implementation also includes a simple rectification to compute a horizontal epipolar line. A self-localization is not part of this master thesis. The system's frame rate depends on the area of interest and lies between 5Hz and 11Hz.

Acknowledgements

Special thanks to:

Gregor Novak
who supported the hardware used.

Robert Sablatnig
for reviewing this work.

My Parents
for their support during my studies.

Contents

Chapter 1

Introduction

> *This is an exercise in fictional science, or science fiction, if you like that better. Not for amusement: science fiction in the service of science. Or just science, if you agree that fiction is a part of it, always was, and always will be as long as our brains are only minuscule fragments of the universe, too small to hold all the facts of the world but not too idle to speculate about them.*
>
> (Valentino Braitenberg, "VEHICLES")

Robots are already present in human society [Dau03]. We tell science fiction stories where robots destroy the world[1] or we just use them to clean the pool [SZCL00]. The spectrum of different robot types ranges from simple plastic dolls, industrial machines, software programmes in the Internet as well as robots which are driving on other planets like the *Mars Pathfinder*[2]. The robot discussed in this thesis belongs to the group of autonomous robot [NM05].
Before an autonomous robot can interact intelligently in its environment it has to sense its surrounding area, because sensing is a key requirement for all but the simplest mobile behavior [DJ00]. One common way to do this is the use of optical sensors as in [WJ04], [BAS+06], [BFD05] and [SWA+02]. [DJ00] writes in his book about vision on mobile robots:

> *Given that we see to navigate effortlessly with vision, it seems natural to consider vision as a sensor for mobile robots.*
>
> [DJ00]

[1] *Daleks* are famous robots characters (mutants in mechanical shells) in a BBC science fiction series named *Doctor Who* which try to "EX-TER-MI-NATE" other life forms and destroy our world.

[2] The Mars Pathfinder was launched on December 4, 1996, by NASA and landed July 4, 1997, on the surface of Mars [http://www.nasa.org].

This master thesis discusses the methods and the implementation of a vision sensor with a stereo camera set for an autonomous soccer-playing robot named *Tinyphoon*. Figure 1.1 shows two Tinyphoon robots with a vision sensor mounted on top. The two cameras of the vision sensor are placed nearly parallel in the aluminium chassis under the vision board. Those cameras produce a considerable amount of data which lies in the standard resolution of 320x240 pixels at 300Kbyte per image pair. The problem is how to reduce this amount of data to a usable set of information to represent the context of scene in the image. Additionally the reduction of information must be done within a time limit so that the robot can react to changes in the information.
The presented vision sensor reduces the information received from the images to a set of detected objects in the soccer environment. Objects like game ball, robots and the goals are detected by the vision sensor. These objects are first detected by their color and then measured by shape. Hence, 3D lines are detected with a feature-based stereo vision algorithm. Those 3D lines can be used to recognize landmarks on the playing field. This set of information (objects and lines) allows the Tinyphoon robot to interact autonomously on a robot soccer playing field. All computation necessary for the detection is done on the embedded hardware on the robot. Therefore, the mechanical size of the vision sensor fits into the fundamental part of the robot which is 7.5cm×7.5cm. The frame rate of the system lies between 5Hz and 20Hz.
The following section describes the motivation behind this thesis. Section 1.2 gives an overview of the robot and the role of the vision system as a unit on the robot. The requirements on the vision systems with the chosen solution are described briefly in Section 1.3. Current related work is presented in Section 1.4. A section about the objective of the thesis followed by the contributions to the scientific community concludes this chapter.

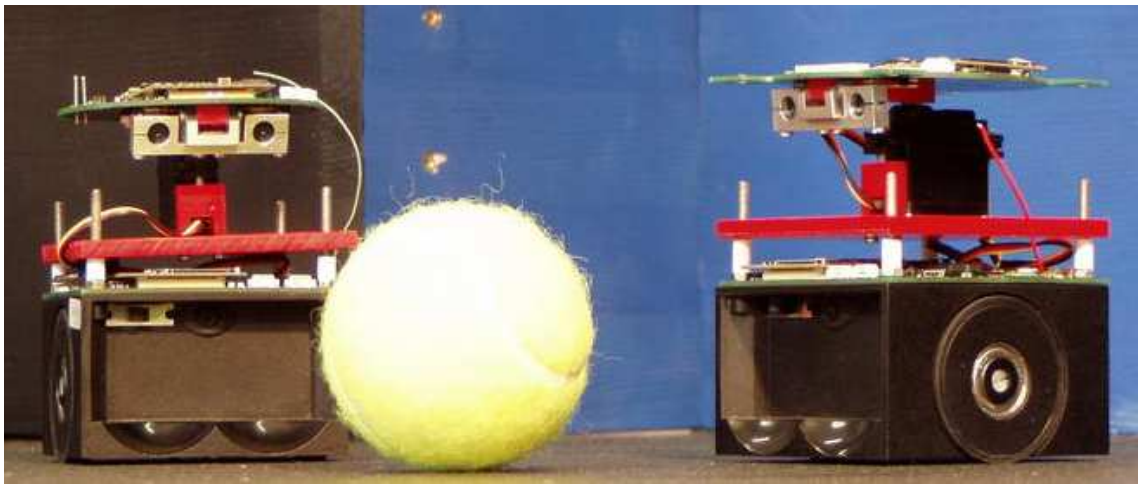


Figure 1.1: The Tinyphoon robot.

1.1 Motivation

FIRA[3] and *RoboCup*[4] are two organizations intended to promote robot soccer. They attempt to provide a standard problem where wide ranges of technologies must be combined and integrated. Robot soccer builds one of these problem domains. Solutions which are designed for such problem domains are also of interest to the current industry and scientific community. This can be seen in the countless published papers and books on this topic. Another non neglectable point is the fun joy working in a team to create a robot.
The two robot soccer organizations can be distinguished by their famous leagues. FIRA is famous for the *MiroSot League* where small robots in a size of 7.5cm×7.5cm play remotely controlled by a centralized strategy with cameras above the playing field. RoboCup's most famous league is the *Four-Legged League*. Figure 1.2 shows on the left side a MiroSot league and on the right side a Four-Legged League playing. The Four-Legged League uses commercial robots like the *Sony Aibo*[5] while the MiroSot robots are designed by their teams. FIRA and RoboCup claim that in the future there will be a league where humans

[Taken from http://www.fira.net]
(a) MiroSot league (FIRA)

[Taken from http://www.robocup2006.org]
(b) Four-Legged League (RoboCup)

Figure 1.2: Two different leagues of different organizations.

will play against robots. But there are currently only a few leagues in which robots play autonomously.

- Four-Legged League (RoboCup)
- Mid-Size League (RoboCup)
- RoboSot League (FIRA)

[3]Federation of International Robot-soccer Association, founded in 1997. Details can be found at www.fira.net.
[4]RoboCup Federation, founded in 1993. Details can be found at www.robocup.org.
[5]Details about the Sony Aibo robot can be found at http://www.sony.com.

- Humanoid League (RoboCup and FIRA)

- Simulation League (RoboCup and FIRA)

Discussions are going on about playing in the Small-Size league of RoboCup and in the MiroSot league of FIRA autonomously. The benefit would be that the development of a more realistic hardware in terms of energy consumption and size will be driven. The new FIRA league will be named AMiroSot (autonomous MiroSot). [KDB+06] describes possible hardware and rules for an AMiroSot league. The motivation in the presented work is to implement a vision system algorithm based on the current Tinyphoon to prepare the robot for AMiroSot. Of course there is not only the vision system present on the robot. The other units must also be designed, integrated and tested. The next section will present the different units on the robot and their purposes.

1.2 System Overview / The Robot

This section covers briefly the role of the vision unit on the robot and gives the reader an overview of the units integrated on the robot. A detailed description of the hardware and the communication and relation between them can be found in Section 5.1.
The Tinyphoon robot has in its current version three units [NRB+06].

Motion Unit: Controls the wheels and collects information from sensors like gyro, acceleration and magnetic field sensors.

Strategy - WMR (World Model Repository) - Reasoning Unit: This unit represents the *brain* of the robot, decisions of what the robot should do next based on the sensor inputs, and the history of the last actions made here. This unit will be named in the following text *Strategy Unit*.

Vision Unit: Detects objects on the playing field by using two CCD (charge-coupled device) cameras.

All three units must interact so that the robot can fulfill its goal, which is to play robot soccer. Based on the sensor information from the motion unit and the vision unit, the strategy unit decides to which position the robot should move next. This position is submitted to the motion unit which sees that the robot reaches the coordinates received. The task of the vision sensor is to extract 3D information of simple geometric structures which is related to the information desired. Those geometric structures are 3D lines related to the landmarks on the playing field, spheres related to the game ball and rectangles related to the goals and the robots.
The dataflow of the implemented vision system is shown in Figure 1.3. Two images are taken at the same time and processed in parallel. Every core rectifies its image, segments it and performs an edge detection on it. The detected blobs are then used to localize structures in the image of related known objects. A blob in the color of the game ball is

used to find the circle related to the sphere. The exact position and size of a structure is computed by using a shape detection algorithm. (Example: A yellow blob represents the game ball therefore a circle shape detection will be applied in and around the blob). The implementation in this thesis does not include a rectangle shape detection but it provides a general interface to detect objects which can easily be enhanced to different types of shape detections. A circle shape detection is currently the only one implemented. Rectangle shapes are detected only by their color. All detected objects are verified by using the detected objects from the second image as reference.
The system also provides 3D lines for a localization estimation. These lines are generated by a feature-based stereo algorithm. The location estimation itself is not part of this thesis.
The Tinyphoon robot is able to move at 3 m/s but at the moment there are no vision systems on the market which are small enough and capable of processing the image information fast enough to travel controlled at such a speed. Even the work presented in this master thesis will only work for a traveling speed around 0.15m/s.

1.3 Requirements and Solutions

The requirements are classified into functional and non-functional requirements. The functional requirements describe the obvious problems of the system fulfilling its task. The non-functional requirements specify criteria to judge the operation of the system.

1.3.1 Functional Requirements

The vision system is designed for a mobile autonomous robot which is able play robot soccer in the AMiroSot League. Therefore the system must be capable of:

Detecting spherical, colored objects to recognize the game ball.

Detecting rectangular, colored objects to recognize players and goal.

Detecting 3D lines to recognize the landmarks on the ground for self-localization.

1.3.2 Non-Functional Requirements

The robot travel speed depends strongly on the vision system speed and its object detection rate. The quality of the system can therefore be measured by:

Frame rate and/or object detection rate: because tests and comparison [BAS+06] to other systems showed that a frame rate between 5Hz and 15Hz is necessary to play robot soccer.

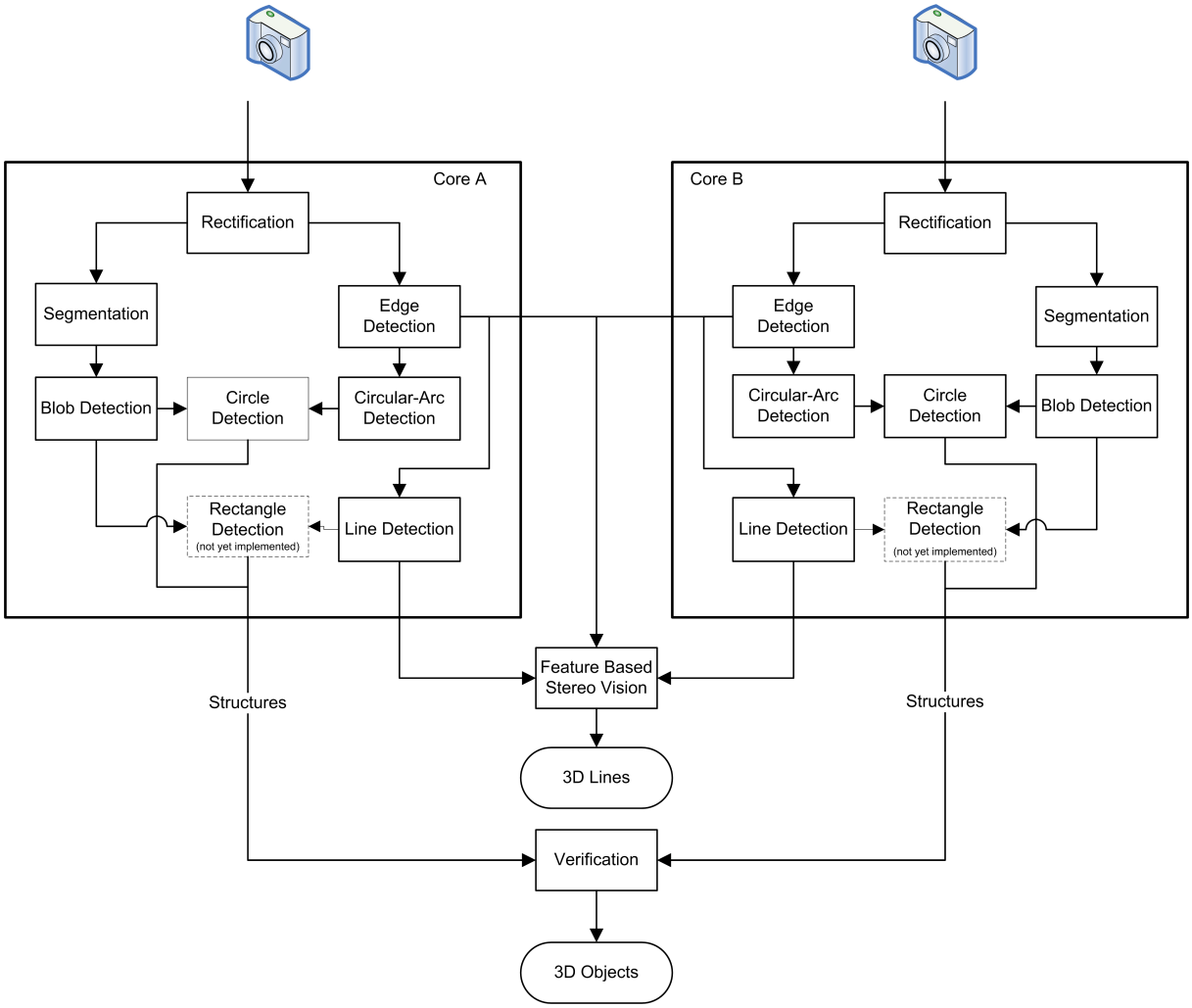


Figure 1.3: Overview of the dataflow on the vision system.

Accuracy: The system should give at least an indication of how successfully a seen object was detected.

Insensible to brightness changes: Even if the rules of the AMiroSot league propose strict conditions for the environment. The system should be insensitive to lighting changes. Experiences from the RoboWorld[6] showed that the audience and the other playing fields next to the environment disturb the lighting conditions during the game.

Simple to use: The vision system should have a clearly defined interface so that it can be used on different robots or machines.

Size, energy and costs are also critical criteria. But the hardware is already given and

[6] 11th FIRA RoboWorld Cup in Dortmund 30th July - 3rd June 2006.

those properties can only be changed minimally by the implementation in this master thesis.

Problems with the frame rate are caused in the DSP (Digital Signal Processor) controller used. Table 1.1 shows a comparison of the DSP[7] used in the implementation and a current state-of-the-art x86 processor[8]. We see that the performance of the DSP is at least five times slower as the performance of the x86 processor. Additionally, the DSP has no floating point unit which causes, for every function call like `tan()`, `tan2()` or `sqrt()`, extra processor cycles to emulate a floating point operation with integer operations.

Processor	x86	DSP	$\frac{x86}{DSP}$
Processor clock rate (Access speed to the cache)	2660 MHz	600 MHz	x4.4
Cache	2048 Kbyte	256 Kbyte	x8
Front Side Bus clock rate (Access speed to the SDRAM)	1066 MHz	133 MHz	x8
SDRAM	>512 MB	32 MB	x16
Power consumption	≈ 65 Watt	≈ 2 Watt	x32
Floating points unit	yes	no	

Table 1.1: Comparison of a x86 processor with the DSP used in the implementation.

1.3.3 Solutions to Functional Requirements

Image processing algorithms are already available in libraries like *OpenCV*[9] but these libraries are designed to work on a x86 platform. The vision unit of the Tinyphoon robot is not based on a PC. It uses instead a DSP Core Module which has low energy consumption around 2 Watts [MON04] and a physical size of 36x32mm.
The functional requirements are fulfilled using the following techniques.

The Hough-transformation for circles to detect the game ball.

A segmentation algorithm like Rosenfeld [RP66] to find colored objects

A line detection based on local and global parameters [10] to find lines.

A feature-based stereo vision to estimate the 3D location of detected lines.

Alternative techniques which were not used in the implementation because of their computation and memory intensive behaviors are:

[7] Analog Devices Blackfin ADSP-BF561 Dual Core DSP 600Mhz [Ana05].
[8] Intel Core 2 Duo Processor E6700 2.66GHz [Int06].
[9] OpenCV is an open-source computer vision library developed by Intel.
[10] Local and global line parameters are described in Section 2.2.1.

A Hough-transformation for lines [DH72] to detect lines.

A SVM (Support Vector Machine) [LW04] to classify/segment the images.

1.3.4 Solutions to Non-Functional Requirements

The following list describes strategies to solve the non-functional problems discussed in Section 1.3.2.

A higher frame rate can be reached by taking care of the processor's resources and the listed strategies.

- *Avoiding slow operations*
  Exact results of floating point operations are in cases where results are used in comparisons not needed. Example: In the case that the longest vector should be selected, the actual length is not of interest. Therefore a comparison of two vectors could be substituted with (1.1), where no floating point operation is needed.
  $$\sqrt{a_0^2 + b_0^2} < \sqrt{a_1^2 + b_1^2} \equiv (|a_0^2| + |b_0^2|) < (|a_1^2| + |b_1^2|) \tag{1.1}$$

- *Simplification*
  Image distortions and other inaccurate measurements during a detection algorithm are coursing tolerances. Floating numbers can be too precise so that the normal C/C++ floating point operation `sqrt()` can be replaced by a fix point square root function based on longhand division [Tur94]. The radian which is a *SI*[11] *supplementary unit* could also be transformed to a system with a domain of 8-bit values. This allows fast comparisons and saves memory. Example: An angle could be represented with a value between `0x00h` and `0xFFh` instead of between $-\pi$ and $+\pi$.

- *Pre-computation*
  The use of a lookup table reduces the cycles for a complex mathematical operation for the cycles used for memory access. A domain of discreet values for angles with integers instead of floating point numbers would benefit from such tables.

- *Image processing algorithms*
  The image processing algorithms have to be designed so that random SDRAM accesses are minimized. This can be achieved by using DMA controllers and optimized data structures. Functions which access the same data for processing like image segmentation and edge detection can be merged into one function to save additional SDRAM accesses.

[11] The International System of Units (abbreviated SI from the French language name Système International d'Unités).

Detection on different levels. Objects should be detected by their color and structure. This also includes an accuracy indicator which gives points for every positive test.

The YUV-Color format [12] separates the chroma and the luma channel. [BAS+06] showed successful results on different lightning contrition with this format.

1.4 State of the Art

Similar applications can primarily be found in the field of robot soccer, because the need to detect objects on the playing field is essential for robot soccer. But a combination of a ball detection system and a stereo system was only proposed in two papers, [NBM04] and [SPV05]. [NBM04] describes an embedded ball detection stereo system where the two extracted spheres from the left and the right image are primarily used to raise the accuracy of the detection. The application in [NBM04] was used for robot soccer. A similar but more complex implementation is proposed in [SPV05] where the system is based on a PC and not for robot soccer.
Most robots of the FIRA Middle Size League are using omnivision cameras to find the objects on the playing field and to localize themselves [YYL05]. But the middle size league robots are big enough to hold a PC for processing on the robot. There are also reports of systems with a combination of an omnivision camera and a "normal" camera in the front [KYY05]. The use of an omnivision system for the Tinyphoon robot is not an option because it would be to large. The related work can be split into two main groups: stereo vision and ball-detection applications.

1.4.1 Stereo Vision Applications

Most embedded stereo implementations take advantage of two programmable chip types, *Field-Programmable Gate Arrays (FPGAs)* and/or DSPs. One of the first reported stereo systems was developed at IRNIA [13] and it was implemented for both DSP and FPGA hardware in 1993 [BBH03]. The system worked with an image size of 256x256 and was able to compute a frame rate of 3.6 *frames per second* (fps). Darabiha proposed in [DRM03] a vision hardware based on an FPGA which is able to compute 30 frames per second at a resolution of 256 x 360 pixel. It also produces depth maps like [CD97]. Unlike [DRM03] and [CD97], [Kon97] used a DSP where he reached a frame rate of 8 fps at 180 x 160 pixel but the system included also a kind of calibration. The most equivalent implementation is reported by [Ent05]. He used a feature-based stereo vision but implemented it on a PC. Reports of more stereo systems can be found in [Kon97] and [BBH03]. [LB03] and [IYT92] propose stereo vision systems with omnivision cameras of different types.

[12] The YUV-Color format will be discussed in Section 2.3.4.
[13] The French National Institute for research in computer science and control

1.4.2 Ball Detection Applications

A usual strategy for finding objects is to find a color[14] blob[15] related to the object's color. [KYY05] used such a blob detection on his middle-size robot to recognize the ball with an omnivision camera connected to a PC. [JD02] also used the ball color but he designed a smaller system integrated to a PDA with a frame rate of 2.5-3.5Hz and the PDA was also used to control the robot. [RRN02] implemented his system based on the object color on embedded hardware and reached a frame rate of up to 16Hz but with low accuracy. [MON04] used hardware similar to the one in this thesis and reached a frame rate of 60Hz but also with low accuracy. [BAS+06] used the same hardware with a single-core Blackfin DSP CM-BF533 in his implementation to integrate a system with higher accuracy with a frame rate between 5Hz and 20Hz. The object color was used to get a rough idea in which area the shape detection should be applied to measure the exact dimensions of the detected object.

1.5 Objective of This Thesis

The goal of this thesis is to design a feature-based stereo algorithm and a ball detection algorithm for given vision hardware on the Tinyphoon robot. The algorithm should take care of the processor's architecture to reach a frame rate high enough to play robot soccer. A frame rate between 5 and 15Hz depending on the processed image size should be reached.

1.6 Contributions

The contributions for the scientific community are firstly in the field of electronic engineering, and secondly in the field of computer science.
The proposed work proves that a low energy consuming (2Watts) embedded stereo vision hardware can be created on a small space (75mmx75mm). This is of interest for the electronic engineering community.
The contribution to the computer science community is the development of:

- A scan line based canny edge detector for embedded systems with DMA-Controllers which works without floating point operations.
- A memory optimized line detection which works in linear time.
- A general concept of detecting known objects in images based on color and shape.

[14] Color includes, in this case black, and white.
[15] Blobs are described in Section 2.3.5.

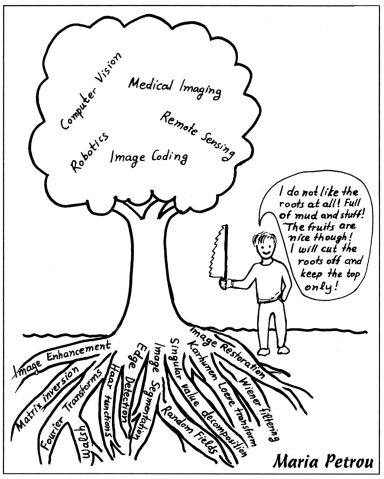
- A new filter to detect edges related to colored circles. This filter optimizes the Hough transformation for circles.

1.7 Structure

Chapter 2 presents fundamental concepts of image processing and feature extraction which are used in implementation. Camera properties and calibration techniques are discussed in Chapter 3. The estimation of distances to seen objects or lines are described with different techniques in Chapter 4. A detailed description of the hardware used and the software implementation is described in Chapter 5. Results to experiments on the developed algorithm are presented in Chapter 6 followed by the conclusion with a discussion.

Chapter 2

Image Features



[Figure taken from [PB99]]

Image features are described in [TV98] as special parts of an image which correspond to interesting elements of the scene. Therefore, the purpose of the vision system defines which elements are interesting. The purpose of our vision system is to recognize elements related to objects from a robot soccer environment in the image and to estimate their location in a second step. This chapter will cover the theoretical background to allow the reader to understand the approaches used to terminate the position of the projection from the game ball, the goals and the landmarks in the image. The estimation of the location of an object on the playing field will be discussed in Chapter 4.
The basic idea of detecting interesting elements in the implementation is to first determinate the location of the element in the image by its color and to then verify and/or measure it by its shape. Therefore, the chapter discusses detection methods of detecting shapes like lines and circular arcs and techniques to segment images in colored areas.
The chapter starts with different edge detectors followed by curve detectors to find lines and circular arcs. Section 2.3 describes techniques to segment images and how to deal with

noise by segmentation. The description of the YUV-Color format and a blob detection algorithm concludes this chapter.

2.1 Edges

Image features can be ordered in a hierarchy. On a basic level we have edges, and based on these we have geometric structures like lines or curves, followed by objects and so on. This section will focus on the extraction of edge elements in images.
Edge elements, or edges, can be described as areas in the image where the pixel value undergoes a sharp change in its surrounding neighborhood. The properties of an edge element are described in the following list and Figure 2.1.

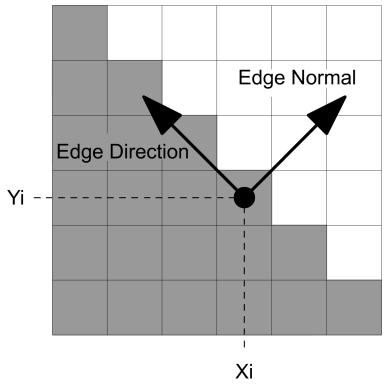


Figure 2.1: Edges and the properties of an edge.

- Edge Position
  Position of the edge in the image with x and y coordinates
- Edge Normal (Edge Gradient)
  A vector looking towards the intensity change.
- Edge Direction
  A vector perpendicular to the edge normal.

The detection of edges works in two steps: first an edge detection, which is then followed by the edge localization. The edge detection builds the first derivation over an image area. The desired edge is localized where the first derivation forms a peak or by the zero crossing of the second derivative. These two steps are necessary because we are dealing with natural images. A brightness change of an edge of interest does not occur between two pixels (Figure 2.2.a). Instead, the change occurs smoothly over many pixels. Figure 2.2.b shows the first derivation and 2.2.c the zero crossings of the second derivation.

There are many algorithms known to detect edges in images like *Sobel, Shift & Difference, Frei & Chen, Roberts, Perwitt and Laplace Enhancer*. Details and more methods can be found in [Bax94, PB99, TV98].

2.1.1 Sobel Operator

The Sobel Operator approximates the intensity changes $\frac{\partial I}{\partial x}$ and $\frac{\partial I}{\partial y}$ by applying two different operators s_x and s_y on an image area a (2.1),(2.2). The partial derivation makes it possible to compute the edge gradient ∇I and edge direction (2.3), (2.4). The actual edge location can be estimated by applying a nonmaximum suppression described in Section 2.1.4.

$$s_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \quad , \quad s_y = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix} \tag{2.1}$$

$$\Delta I_x = \sum s_x \cdot a \approx \tfrac{\partial I}{\partial x} \quad , \quad \Delta I_y = \sum s_x \cdot a \approx \tfrac{\partial I}{\partial y} \tag{2.2}$$

$$|\nabla I| = \sqrt{\Delta I_x^2 + \Delta I_y^2} \tag{2.3}$$

$$\nabla I_\alpha = \arctan 2 \left(\Delta I_y, \Delta I_x\right) \tag{2.4}$$

The computation of the partial derivatives with the operator can be simplified in the case of the Sobel operator. The symmetry in the operator allows us to store the result of one multiplication of one row (s_y) and/or column (s_x) to reuse it only by changing the sign to increase the performance.

2.1.2 Frei & Chen Operator

Other similar gradient-based solutions like the Sobel operator are using different operators. Frei & Chen has similar operators (2.5) like the Sobel, but the $\sqrt{2}$ allows a better gradient approximation with the cost of a higher computational complexity.

$$s_x = \begin{pmatrix} -1 & 0 & 1 \\ -\sqrt{2} & 0 & \sqrt{2} \\ -1 & 0 & 1 \end{pmatrix} \quad , \quad s_y = \begin{pmatrix} 1 & \sqrt{2} & 1 \\ 0 & 0 & 0 \\ -1 & -\sqrt{2} & -1 \end{pmatrix} \tag{2.5}$$

2.1.3 Laplace Operator

Unlike the operators named before, the Laplace operator $\nabla^2 f$ (2.6) directly computes the second derivation. Because of that, the edge direction gets lost.

$$\nabla^2 f = \langle \frac{\partial^2 f}{\partial x^2}, \frac{\partial^2 f}{\partial y^2} \rangle \tag{2.6}$$

The implementation of a Laplace operator can be done by a kernel with the size of $m \times m$. Such a kernel with the size of 3×3 is shown in (2.7).

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix} \tag{2.7}$$

Unlike other operators like Sobel, this operator is an omnidirectional operator. This means there is only one operator necessary to find edges, while on the Sobel, Frei & Chen or Roberts at least two operators are involved.
It is notable that the Laplace operator is sensitive to noise. A combination between a Gaussian (2.8) and the Laplace operation can work against this problem and is named *Laplacian of Gaussian* Filter (LoG) (2.9). This function is also known by the name *Mexican Head*.

$$g(x, y, \sigma) = \exp -\frac{x^2 + y^2}{2\sigma^2} \tag{2.8}$$

$$LoG(x, y, \sigma) = \left(\frac{x^2 + y^2 - 2\sigma^2}{\sigma^4}\right) \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right) \tag{2.9}$$

2.1.4 Canny Edge Detector

A combination of filters and operations to extract edges is the *Canny Edge Detector* [Can86] which uses a Gauss operator, a Sobel operator, a nonmaximum suppression and a hysteresis threshold. Figure 2.2 shows all major steps in the canny edge detector.

1. The image will be smoothed by applying a Gauss operator with zero mean and standard derivation. This is necessary because the following Sobel operator is sensible to noise [PB99].
2. By applying the Sobel edge operator, a gradient vector ∇I can be computed for every pixel looking toward the brightness change.
3. An edge with a gradient length $|\nabla I|$ under a threshold τ_e will be removed.
4. The *nonmaximum suppression* eliminates edges where at least one neighbor in the edge direction has a bigger gradient length than $|\nabla I|$.
5. The *hysteresis threshold* groups connected edges[1] with a $|\nabla I| > \tau_l$ and removes a group if there is no edge in the group with $|\nabla I| > \tau_h$.

[1] Connected edges will be described in Section 2.3.2.

(a) Source (b) First derivation (d) Second derivation (c) Hysteresis threshold

Figure 2.2: Canny edge detection in different processing stages.

2.1.5 Summary

Natural images taken by cameras are noisy because of imperfect fabrications of camera lenses and the optical sensor. According to [PB99], the canny edge filter is the best choice for natural images. The Gaussian parameter σ should be taken as a free parameter estimated by using experimental results. The output of the canny edge detector can be used as input for the following described curve and shape detection to find lines and circles in images.

2.2 Curves and Shapes

Detected edges provide the foundation for the methods in this section. Curves and shapes are directly related to the underlying edges in the image. This section is divided into two parts. The first deals with straight lines and the next one with circles followed by a brief summary.

2.2.1 Straight Lines

If we are speaking about lines we have to distinguish between lines and line segments. A line is an infinitely thin, infinitely long, perfectly straight curve [Ste02]. A line can be represented in a two-dimensional Cartesian coordinate system in the so called *Slope-Intercept Form* with slope m and y-intercept b shown in the equation (2.10).

$$y = mx + b \tag{2.10}$$

A line segment is a part of a line that is bounded by two end points and contains every point on the line between its end points [Ste02]. Normally a line segment is described by two points.
The presented strategies to extract lines can be grouped into two classes:

- Edge transformation or global parameters
  Edge properties are transformed into a *Parameter Space* where the space axes represent the properties of a line. Every point in such a space represents a line. A

maximum in the parameter space represents a potential candidate for a line in the image coordinate system. [DH72] was one of the first to document a line detection based on Hough transformation, which is named after the related 1962 patent of Paul Hough.

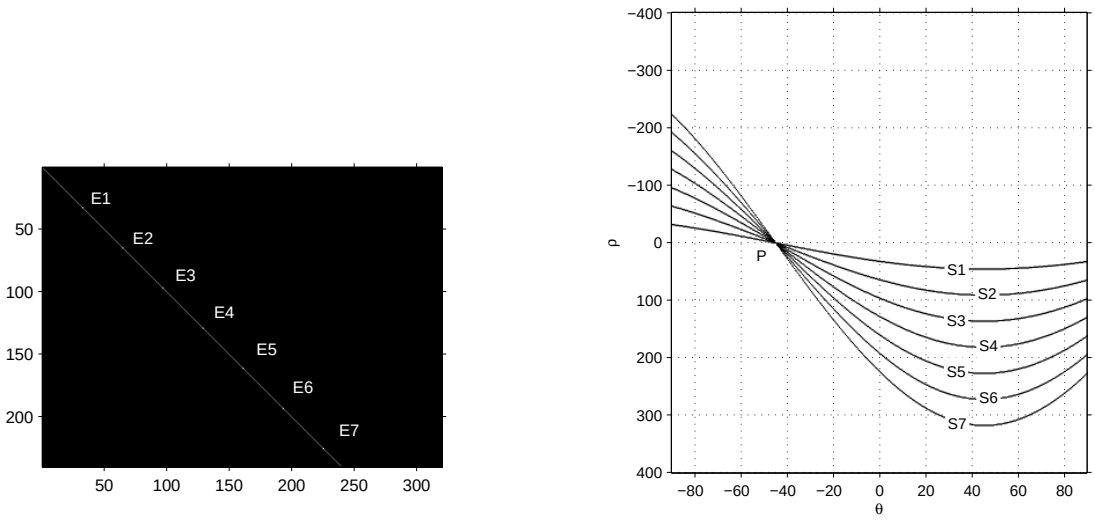
- Edge grouping or local parameters
  Edges on a line have shared characteristics including the gradient direction. If connected edges with similar characteristics are grouped together, statistical methods can be used to determinate if such a group forms a line segment. [Hor73, BHR86, CB03, Ent05] are using this advantage in their line detectors.

Hough Transformation

The Hough transformation (HT) describes a way to detect lines and simple curves in binary images. The gradient direction or the edge strength is not needed to perform this algorithm. The detection of the line can be performed in two steps. The input is an edge image which can be generated by a canny edge detector.

1. Transformation
   Every edge $E_(x_{img}, y_{img})$ in the image represents a curve in the parameter space. The parameter space represents a system where the axes are the properties of the curve to detect, in our case, the properties of a line. As we know, a line can be represented in different ways, therefore the parameter space is different for every representation. A simple way to represent a line is by its slope m and y-intercept b in slope-intercept form (2.10). The curve of an edge in the $< m, b >$ parameter space will be in this case a line L (2.11).
   $$L = \{(m, b) | y_{img} = m x_{img} + b\} \tag{2.11}$$
   The curve would be a sinusoid S (2.12) if we would use the properties ρ and θ for the $< \rho, \theta >$ parametric space of a line in the polar system (Figure 2.3). ρ stands for the distance and θ for the angle of the edge $E_(x_{img}, y_{img})$ to the origin.
   $$S = \{(\rho, \theta) | \rho = x_{img} cos(\theta) + y_{img} sin(\theta)\} \tag{2.12}$$
   Figure 2.11 shows a source image with edges and the related $< \rho, \theta >$ parameter space.

2. Peak search
   An intersection of two or more curves on one point P occurs in the Hough space if the related edges are on the same line in the image. A peak will be the result of a line in the parameter space and this peak can be detected by using different algorithms like in [Dav92]. The transformation back to the image coordinates depends on the chosen parametric space.



(a) Source image (b) Parameter space

Figure 2.3: The left image shows a line with a slope of $m = 1$ and $b = 0$. The right image represents a parameter space with ρ and θ as axes. The sinusoids $S1 - S7$ are the related curves to the edges $E1 - E7$ in the source image. $P_{45°,0}$ points to the intersection which describes the line of the source image in polar coordinates.

A line can have infinite plus or minus m and b values. Therefore a $< m, b >$ parameter space deals with the risk of loosing intersections. This is the reason why most implementations use the $< \rho, \theta >$ parameters space.
Another finite parameter space is the one used by the *Muff Transformation*. It uses the two points where the line would intersect with the rectangle image plane (Figure 2.4).
Normally the detection of a line is not sufficient. We attempt to find the line segment on the line which matches the line segment shown in the image by the edges. This can be detected by searching the last occurrence of an edge on the detected line. Algorithms to terminate the endpoints of a line are proposed in [TV98] and are not further described here.

Burns

The Burns algorithm groups edges by their gradient orientation in so-called *line-support regions*. The actual localization of the line itself is then terminated by using the gradient strengths of the edges in the line-support-regions. Unlike other papers, [BHR86] defines a line with a start and an endpoint like a line segment but he also adds attributes like a width and a contrast to the line. The input of the Burns algorithm is an edge image with all gradient information of every pixel such us the output of a Sobel edge detector. The algorithm can be described in the following steps.

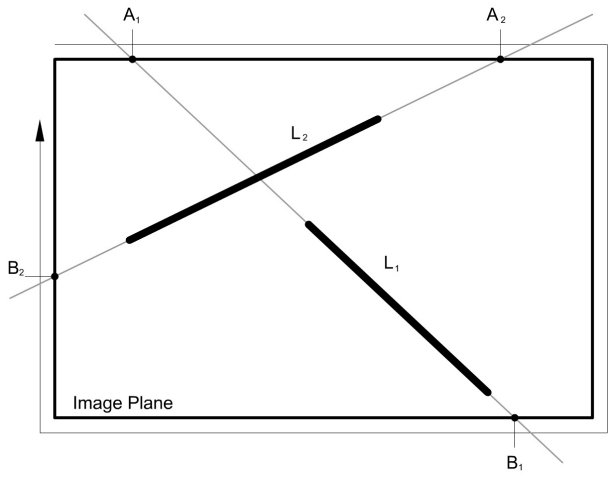


Figure 2.4: Line properties of the muff transformation. A line L_i will be described by two Points A_i and B_i. A and B are presenting the position on the perimeter of the image plane.

1. Grouping edges into line-support-regions by the gradient orientation.
   [BHR86] used predefined intervals with 8 or 16 partitions to classify edges, Figure 2.5.b. He also showed how a grouping can be accomplished using overlapping partitions because a natural line tends to lie across partitions.

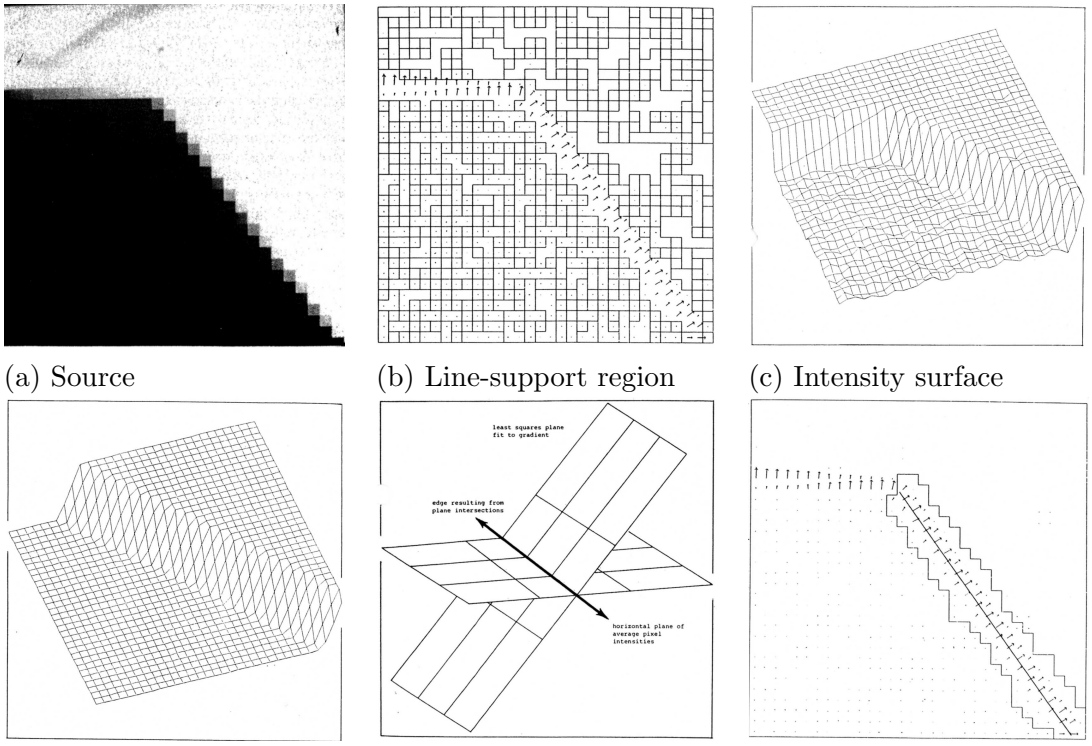
2. Planar fit on the ramp of the change in the intensity surface.
   An approximation of a plane to the ramp of the intensity surface is done in a line-support region. The intensity change in a line-support region can be described by a plane ramp overlayed with noise, shown in Figure 2.5.c. [BHR86] tries to remove the noise in the ramp by using a least-square function and to fit then a plane, Figure 2.5.d.

3. Extraction of the line attributes.
   The line attributes can now be extracted by the location and orientation of the planar fit from the last step as well as the length, contrast and width, Figure 2.5.e. The location and orientation is determined by using a second plane which intersects the planar fit at the average height of the intensity surface, Figure 2.5.f.

4. Filtering lines
   Depending on the user, not all extracted lines are of interest. Texture lines are normally buried in short lines while long lines mostly represent depth edges.

2.2.2 Circles

Extracting circular features are common features for location estimation [SRTSB92]. We have to distinguish between the detection of circular arcs and filled circles. The detection



(a) Source (b) Line-support region (c) Intensity surface

(d) Ramp (e) Planar fit (f) Line

[Figures and comments are taken from [BHR86]]

Figure 2.5: Typical line approximation using the burns algorithm. (a) Source image. (b) Regions produced by a connected-components algorithm with two line-support-regions. (c) Surface plot of an edge-intensity profile. (d) Planar model of the line obtained by a least-squares fit weighted by gradient magnitude. (e) The straight line is obtained by the intersection of a weighted planar fit with a horizontal plane representing the average intensity. (f) The resulting straight line overlaid on the set of pixels making up the line-support region.

of a filled circle can be reduced to the problem of detecting a circular arc but not vice versa. Therefore, blob[2]-based detections for filled circles cannot be applied for circular arcs.

Circular Arcs

The description how to detect edges describing an arc curve is widely circulated ([KBS75], [Kie92], [IHL99]and [BAS+06]). Using the Hough transformation is the most popular technique for detecting such arcs. The first versions [DH72] and [KBS75] used a parametric space with three axis while later versions [IHL99] and [BAS+06] used an optimized algorithm by splitting the problem into two parts. [IHL99] presented in his work the following

[2]Blobs will be described in Section 2.3.

two steps to detect circular arcs.

1. Detection of the circle center
   This is performed by finding a peak caused by the intersection of lines drawn toward the center. The line to the center can be found as a normal to a virtual line between connected edges.

2. Detection of the circle radii.
   The *Radius Histogram*, which represents all involved edges of the last step with their distance to the detected center is used to find the radius.

[BAS+06] used the gradient direction in the first step to define the line direction. This reduced the complexity but made the algorithm sensitive to noise.

Filled Circles

Another popular approach in machine vision for detecting circular, filled shapes is to use the pixel area of a blob. Such detection is fast but not accurate. The accuracy could be increased if the radius of the filled circle would be known. This is normally the case in an automatic visual inspection (AVI) system. We have to be aware that only the area size of a blob will be compared to a circle, therefore other shapes can also fulfill this property of equation (2.13). If P_a is close to one, the probability increases that the pixel area of this blob will be a circle of the radius r.

$$P_a = \frac{(r^2 \times \pi)}{blob.pixelcount} \tag{2.13}$$

2.2.3 Summary

The Hough transform is the widest spread line detection algorithm on PC platforms but the memory consuming parameter space and the read and write access make the Hough transformation on an embedded system slow compared to other local line detectors [CB03]. Even optimized implementations, where the Hough transformation takes care of local properties, are not as good on memory weak-systems as a grouping algorithm. A combination of the Burns algorithm [BHR86] and the local lines algorithm of Climer [CB03] lays the foundation idea for the implementation described in this thesis. The algorithm constructed functions scan-line-based in linear time like [CB03] and uses local features to group edges to lines as in [BHR86]. The algorithm will be described later in Section 5.2.2.

2.3 Image Segmentation

The robot soccer rules define unique colors for the game ball, the goals, the landmarks and the two playing teams. The detection of such uniquely colored areas in the image can be used to recognize objects from the soccer environment. Image segmentation algorithms are techniques to recognize such areas.
Jarvis describes the process of the image segmentation in the following way:

> *Segmentation is the process by which data are grouped into nonoverlapping, meaningful components. In the context of image processing, segmentation concerns the grouping of pixels into disjointed, homogenous regions, which are hopefully consistent with distinguishing the essential objects and/or their components in the image.*
>
> R. A. Jarvis in [BH99] regarding image segmentation

This section focuses on image segmentation for detecting interesting parts of the image by using simple operations to group and distinguish pixels in the image, based on their color. Therefore, this section starts with algorithms on binary images. The first subsection describes the opening and closing operations for removing noise in binary images followed by a description for defining connectivity between pixels. The subsection 2.3.4 discusses the YUV color format used in the implementation and how the previous algorithms for binary images can be extended for color images. Blobs are then described, including how they are used to store the information extracted using the segmentation algorithm. A short summary concludes this section.

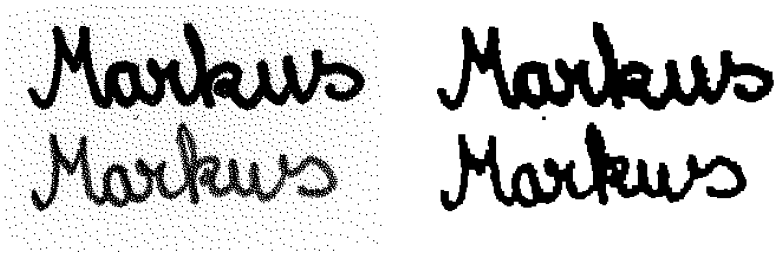
2.3.1 Open and Closing

Let us examine on a binary image where a pixel describes the detection of an interestingly colored object. The occurrence of a single pixel alone does not confirm the existence of the object of interest in the scene. The single pixel can be caused by reflections in the scene or by noise. The probability that the pixel is related to an interesting object increases if the pixel is surrounded by other pixels with the same value. An opening operation removes single-pixel object anomalies by applying an erosion operation (2.14) followed by a dilation operation (2.15). The closing function fills single-pixel gaps and is the inverse sequence of the opening operation, e.g. a dilation followed by an erosion.

$$a_{i,j} = \begin{cases} 1 \text{ , if } a_{i,j} = 1 \text{ and for all neighbors } |_{k=i-1}^{k=i+1}|_{l=j-1}^{l=j+1} a_{k,l} = 1 \\ 0 \text{ , else} \end{cases} \tag{2.14}$$

$$\text{For all neighbors } |_{k=i-1}^{k=i+1}|_{l=j-1}^{l=j+1} a_{k,l} = \begin{cases} 1 \text{ , if } a_{i,j} = 1 \\ 0 \text{ , else} \end{cases} \tag{2.15}$$

Both opening and closing sequences can be applied multiple times to extend the effect. The closing sequence will fill bigger gaps with multiple applications and the opening will remove bigger spikes in the binary image. Figure 2.6 shows a source image with single pixel anomalies on the left side and on the right side the image after a closed and an opened operation where all single pixel anomalies have been removed. Details on this topic can be found in [Bax94].



(f) Source (b) Closed then opened

Figure 2.6: Combination of the dilation and erosion operation sequences.

2.3.2 Connectivity

Before we are able to find connected areas related to an object in the image, we have to define when a pixel is a part of a connected area. The implementation therefore uses an eight-pixel neighborhood to define such areas.
Let us describe the eight-pixel neighborhood using a binary image represented by a $m \times n$ matrix with the entries $a_{i,j}$ with $(0 \leq i \leq m, 0 \leq j \leq n)$. Two pixels $a_{i,j}$ and $a_{k,l}$ are neighbors if $max(|i| - |k|, |j| - |l|) = 1$. Hence, two pixels are connected if they are neighbors and have the same value $a_{i,j} = a_{k,l}$, [RP66]. The next subsection describes how connected components can be detected in images.

2.3.3 Determining Connected Components

[RP66] described in 1966 how to find *connected components* called segments of a binary image in a *scan line based algorithm*. The algorithm scans line by line over the image and decides whether a pixel $a_{i,j}$ belongs to a segment or not. The output is an image where every pixel is labeled with a value corresponding to which connected component the pixel belongs. Only four neighbors of the pixel are necessary to mark the pixel with a label.
The basic steps of the algorithm are as follows:

1. We start with an empty label index in the lookup table and scan every pixel $a_{i,j}$ of the image from the left to the right, line by line. The target image b is also blank at the beginning.
2. If $a_{i,j} > 0$,
   - and no neighbor $b_{i+1,j-1}$, $b_{i,j-1}$, $b_{i-1,j-1}$ and $b_{i-1,j}$ is labeled then label in $b_{i,j}$ with a new label and add this label to the lookup table with equal index and value,
   - and if only one label type occurs in the neighborhood $b_{i+1,j-1}$, $b_{i,j-1}$, $b_{i-1,j-1}$ and $b_{i-1,j}$, then label in $b_{i,j}$ with the same label as the occurrence in the neighborhood, (Figure 2.7.b shows this processing step.)
   - and if two different label types A and B occur in the neighborhood $b_{i+1,j-1}$, $b_{i,j-1}$, $b_{i-1,j-1}$ and $b_{i-1,j}$ then label $b_{i,j}$ with label A and change all occurrences of the label B in the target image from the current and last line to A as well as the occurrence of B to A in the value row of the lookup table. Figure 2.7.b shows this processing step. (Info: Three label types in the neighborhood are not possible.)
3. Rename all labels in the target image b with the corresponding value entry from the lookup table.

Thus far, we have discussed algorithms for binary images in these sections. The next subsection introduces us to the YUV color format and how the segmentation algorithm can be applied to images with color information.

2.3.4 YUV Color Format

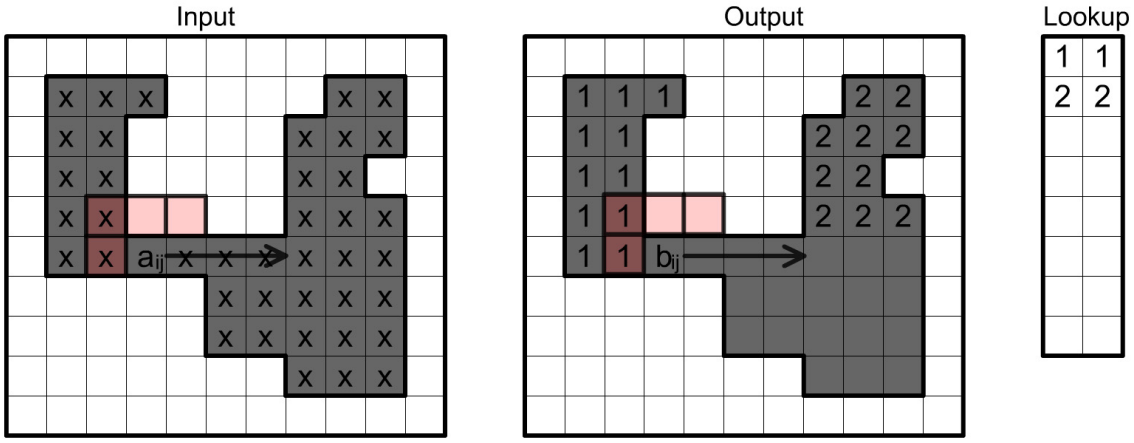
The YUV color format is the format provided by the cameras in the implementation. In brief, a YUV image is the extension of a gray image with two more channels of color information. It was first used to keep old monochrome TVs compatible with the new color TVs. PAL[3] and NTSC[4] use similar YUV Formats. Y represents the luminance of the gray image, U and V the chrominance. But, in fact, it is a little bit more complex, because a human eye is more sensitive to the green colors[5], therefore the transformation from a YUV color format to a RGB format is done with the equation (2.17) instead of (2.16) [NH04]. (2.18) defines the chroma components.

$$Y = R + G + B \tag{2.16}$$

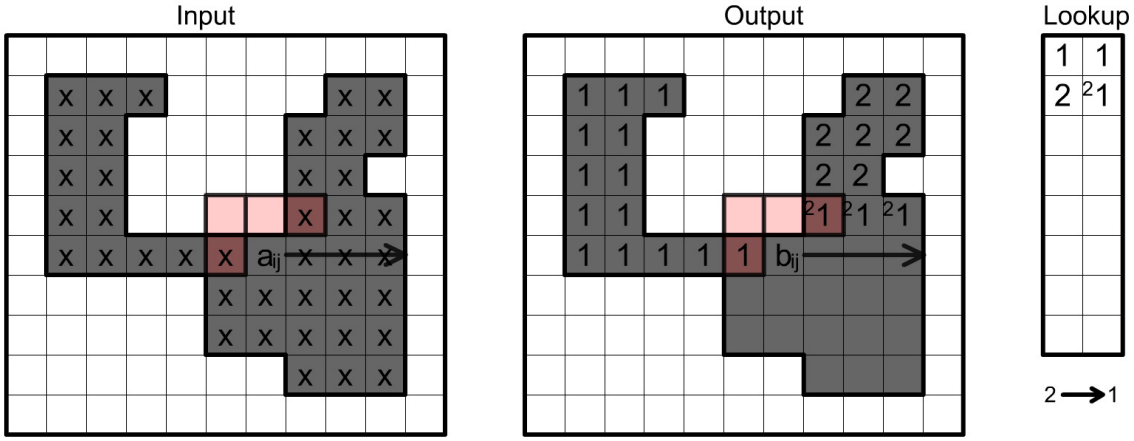
[3]PAL (phase-alternating line) is analog television system in use in Europe except France, Australia, part of Africa, South-America and Asia.

[4]NTSC (National Television System(s) Committee) is the analog television system in use in the United States, Canada, Japan, South Korea, the Philippines, and some other countries, mostly in the Americas

[5]Details to the visible color spectrum of humans can be found at www.wikipedia.com under the keyword "Visible spectrum".



(a) The $b_{i,j}$ will simply be marked with the same label as $b_{i,j-1}$. No changes in the lookup table.



(b) The $b_{i,j}$ had neighbors with different labels. The label 2 was removed from the right side of the lookup table and from the line pixel array $b_{i-1,j}$ to $b_{i,j-1}$. The tiny numbers in the left upper corners show the state before the labels were removed.

Figure 2.7: Two cases of the Rosenfeld segmentation algorithm.

$$Y = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B \tag{2.17}$$

$$\begin{aligned} U &= B - Y \\ V &= R - Y \end{aligned} \tag{2.18}$$

A certain color can therefore be defined as an angle between U and V. *Color segmentation* can be described as a sequential or a parallel execution of methods proposed in Section 2.3.3 on distinct color information. This can be done on the color channels or if a pixel is defined by a numerically distinct number of classes. Figure 2.8 shows segmentation by colors where the segmentation was applied on a red color and a color like brown. Figure 2.8.b shows the connected component detected as gray areas. Blobs are shown as rectangles in Figure 2.8.c which are discussed in the next subsection.

2.3.5 Blobs

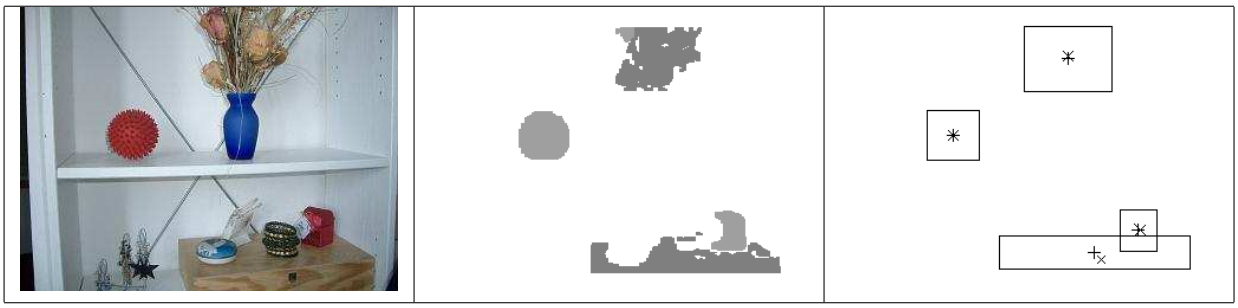
The information extracted by the segmentation algorithm can be stored in blobs. A blob can be seen as a brief description of an image segment. It holds limited information about the related segment, including:

- *blob.rec*, *blob.pos*
  Borders of the segment as a rectangle and its position
- *blob.pixcount*
  Number of pixels in the related segment
- *blob.cc*
  Color Center of Mass (2.19) and (2.20).

$$blob.cc_x = \frac{\Sigma(\text{of all x pixel coodinates})}{blob.pixcount} \tag{2.19}$$

$$blob.cc_y = \frac{\Sigma(\text{of all y pixel coodinates})}{blob.pixcount} \tag{2.20}$$

A threshold is used to cut out segments under and above a certain size. The size of a segment is directly related to the pixel count *blob.pixcount* of a blob. Therefore, thresholds can be used to remove unwanted segments. Figure 2.8.c shows detected blobs where the center of a blob is marked with + and the color center of mass with ×. We can also see that not all segments were used to create blobs because their sizes were below the threshold of the minimum pixel count.



(a) Source (b) Segments (c) Blobs

Figure 2.8: A segmented image with the related blobs.

2.3.6 Summary

This section provided an overview of the topic image segmentation for detecting colored areas in images. We presented the closing and opening operation sequences for removing noise in binary images and the Rosenfeld algorithm to find connected components. The

extension of the segmentation algorithm to the YUV color format was described because this format is then also used in the implementation.
Together with the previous Section 2.2, this chapter represented techniques on how to locate objects like the game ball as a blob in the image and how to measure them by their shape.

Chapter 3

Camera Systems

You have been weighed,
you have been measured,
you have been found wanting

(Brian Helgeland, "A Knight's Tale ")

This chapter describes the geometric constraints of simple pinhole camera systems up to non-coplanar stereo vision systems with calibration techniques. It starts with the geometry of monocular systems followed by their calibration methods. Then stereo systems in a coplanar and a non-coplanar form are described. The chapter ends with a discussion about rectification and a summary.

3.1 Camera Geometry

This section describes the relation between objects in the world and their projection on an image plane.

3.1.1 Pinhole Camera

A pinhole camera is a camera without a lens. The light passes through a small hole and projects an inverted image on a wall behind the hole. Figure 3.1 shows a schematic of a pinhole camera. The sharpness and brightness of the projection depends on the size of the pinhole. A bigger pinhole causes a less sharper but brighter image. If the origin of the camera is placed at the pinhole, the size x of a projected vector on the image plane can be described with the equation (3.1). In the same way, y can be computed (3.2) in three-dimensional space. It can be seen by looking at the schematic view in Figure 3.2

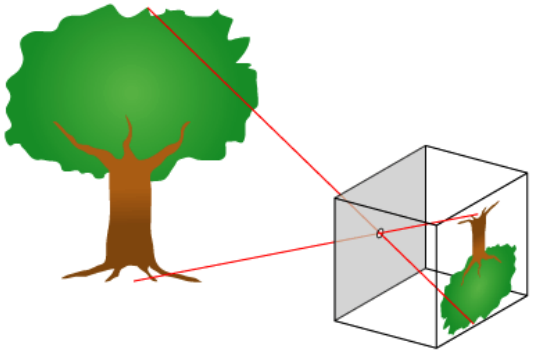


Figure 3.1: Pinhole Camera
[http://en.wikipedia.org/ GNU Free Documentation License]

that a virtual image plane can be drawn in front of the pinhole camera. This image plane, unlike the projection behind the pinhole is not inverted. The non-inverted projection on the virtual image plane is sometimes used to simplify the schematic.

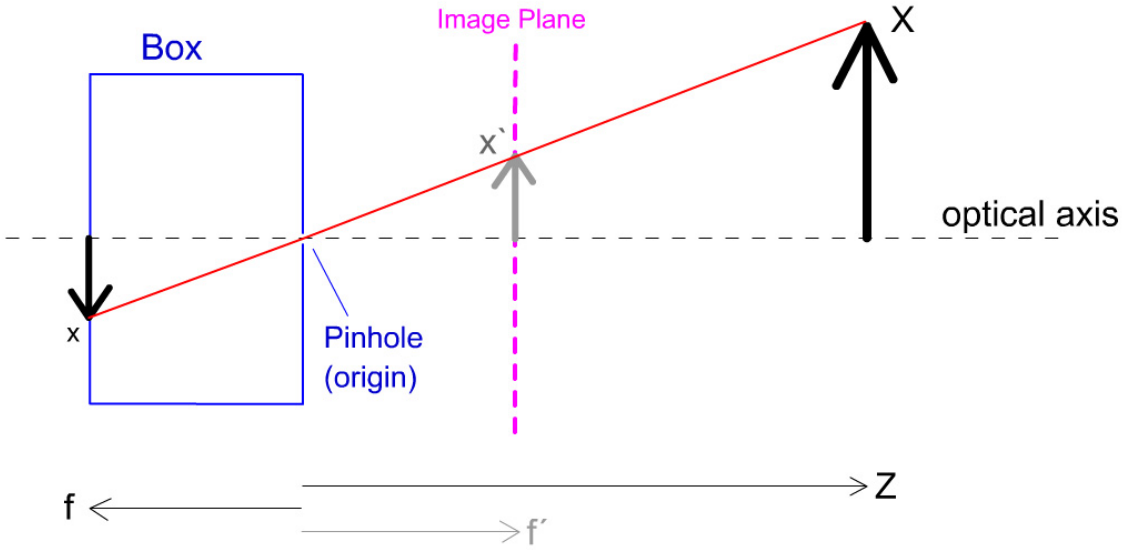


Figure 3.2: Pinhole Camera Scheme

$$x' = -x = f\frac{X}{Z} \tag{3.1}$$

$$y' = -y = f\frac{X}{Z} \tag{3.2}$$

We do know now how to compute where a three-dimensional point $P = (X, Y, Z)$ will be projected on the image plane $p = (x, y)$. But if we assume that the image plane will be

replaced by a CCD chip, we must establish some more properties. Once the CCD chip is placed at its position, it can happen that the optical axis does not pass through the origin of the chip. Hence, we want to represent the point $p = (x, y)$ in pixel units and not in an SI system. For that we must introduce more variables.

- s_x, s_y
  Effective size of a pixel in a horizontal and a vertical direction in an SI prefix (millimeter).
- o_x, o_y
  Coordinates in pixels where the image plane intersects the optical axis.
- x_{img}, y_{img}
  Position of a point p in pixel units.

Now the point $p_{img} = (x_{img}, y_{img})$ in pixels can be computed using the equations (3.3) and (3.4)

$$x_{img} = -\frac{x}{s_x} + o_x \tag{3.3}$$

$$y_{img} = -\frac{y}{s_y} + o_y \tag{3.4}$$

3.1.2 Camera with a Thin Lens

A camera with a thin lens allows more light arrays to enter the camera than one with a pinhole. If we consider a point P not too far from the optical axis, a thin lens will focus all light arrays from P to a point p shown in Figure 3.3. A pinhole camera would only allow light rays to pass through the point O. Because of that, a pinhole camera has an impractically long exposure time.
An image plane placed normal to the optical axis through the point p in a camera model with a lens would show a sharp projection of the point P. If the image plane is placed nearer or further away from the lens, the projection would become blurred, but as long as the blurring is smaller than the resolution of the image sensor, the projection will still be received as a sharp image.
Important equations to compute properties are the *Fundamental Equation for Thin Lenses* described in equation (3.5) and the *Field of View* (3.6). The field of view depends on the usable diameter of the lens d which can vary in its physical size.

$$\frac{1}{\hat{Z}} + \frac{1}{\hat{z}} = \frac{1}{f} \tag{3.5}$$

$$\tan\omega = \frac{d}{2f} \tag{3.6}$$

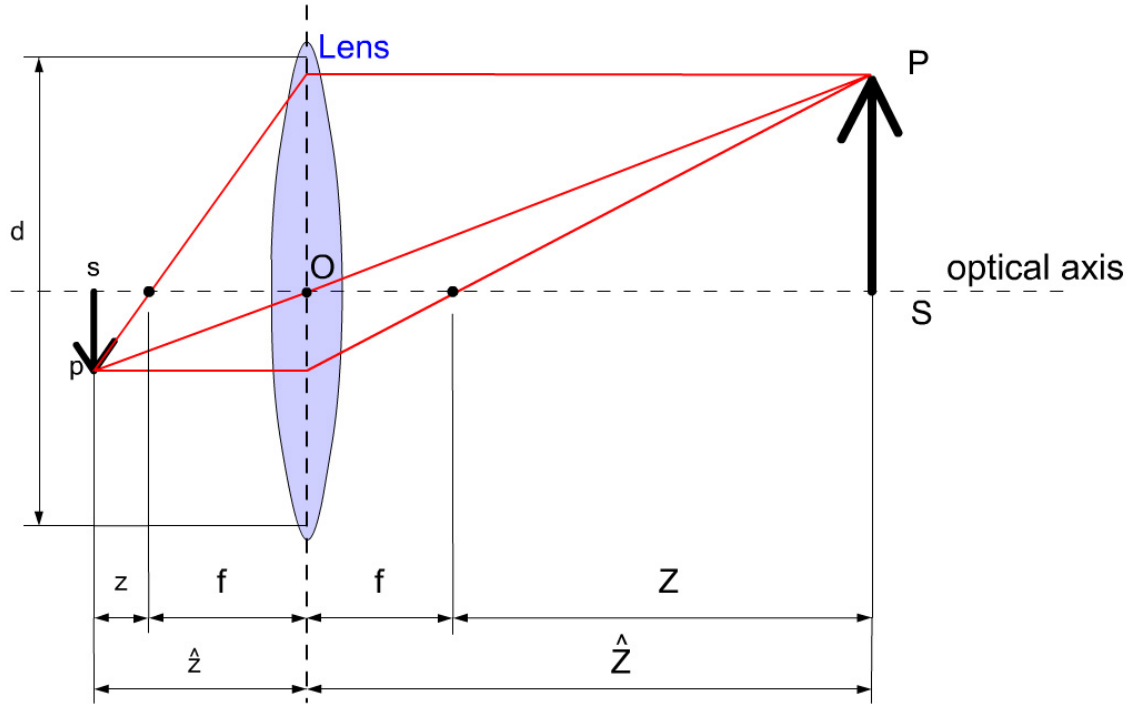


Figure 3.3: Lens Camera Scheme

The use of lenses causes distortions. In particular cameras with a large field of view, strong radial distortions are created. Those distortions become stronger at the border of an image and are non-existent at the image center. κ_1 and κ_2 describe those displacements in the equations (3.5, 3.8 and 3.9) where x_d and y_d are the distorted points and r represents the distance to the center.

$$x = x_d(1 + \kappa_1 r^2 + \kappa_2 r^2) \tag{3.7}$$

$$y = y_d(1 + \kappa_1 r^2 + \kappa_2 r^2) \tag{3.8}$$

$$r^2 = x_d^2 + y_d^2 \tag{3.9}$$

Emanuele Trucco described in his book the use of the distortion variables κ_2 and κ_1 with the following statement:

> *Since they* (κ_2 and κ_1) *are usually very small, radial distortion is ignored whenever high accuracy is not required in regions of the image, or when the peripheral pixels can be discarded. If not, as* $\kappa_1 \ll \kappa_2$, κ_2 *is often set equal to 0, and* κ_1 *is the only intrinsic parameter to be estimated in the radial distortion model.*
>
> [TV98] about κ_2 and κ_1

3.1.3 Intrinsic and Extrinsic Parameter

The intrinsic and extrinsic parameters describe the transformation between the 3D world coordinate system and the 3D camera coordinate system and then to the 2D system on the image plane as shown in Figure 3.4. The translation T and rotation R between the world system and camera system can be realized with a translation and a rotation matrix named *Extrinsic Matrix* M_{ext} (3.10). The *Intrinsic Matrix* (3.11) characterizes the camera geometry with the properties of $f, s_x, s_y, o_x, o_y, \kappa_2$ and κ_1. But as [TV98] already mentioned, the distortions κ_2 and κ_1 are only relevant for high accuracy computations. The following models only describe camera systems without distortions. (3.12) and (3.13) are finally used to compute the 2D coordinates on the image plane.

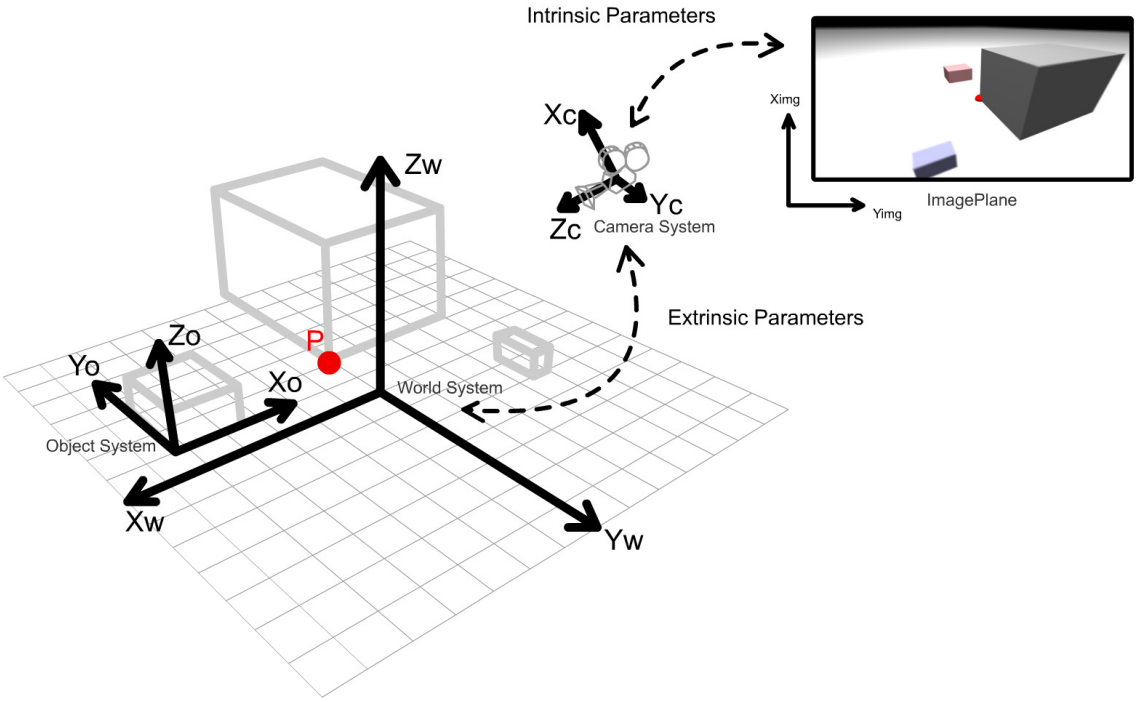


Figure 3.4: Schematic view of intrinsic and extrinsic parameters

$$M_{ext} = \begin{pmatrix} r11 & r12 & r13 & T_x \\ r21 & r22 & r23 & T_y \\ r31 & r32 & r33 & T_z \end{pmatrix} = \begin{pmatrix} R_1^\top & T_x \\ R_2^\top & T_y \\ R_3^\top & T_z \end{pmatrix} \tag{3.10}$$

$$M_{int} = \begin{pmatrix} -f/s_x & 0 & o_x \\ 0 & -f/s_y & o_y \\ 0 & 0 & 1 \end{pmatrix} \tag{3.11}$$

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = M_{int} M_{ext} \begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix} \tag{3.12}$$

$$\begin{aligned} x_{img} &= \tfrac{x_1}{x_3} \\ y_{img} &= \tfrac{x_2}{x_3} \end{aligned} \tag{3.13}$$

3.2 Camera Calibration

The idea of a camera calibration is to find the intrinsic and extrinsic parameters by taking images of scenes where certain properties are known or able to be extracted. [Zha00] divides calibration techniques into two groups.

- Three-dimensional, reference object-based calibration:
  The techniques use a calibration pattern or an object. Depending on the technique and the calibration object, one or multiple images from different known or unknown positions must be taken. Object-based calibration techniques can be very accurate and effective [Zha00] but they suffer from complicated recalibrations if the setup changes.
- Self-calibration:
  A camera with unknown internal parameters takes images in a continues stream of a static scene. A calibration object is not necessary [Har94]. [PKG99] posted in his paper a technique which is also able to deal with uncalibrated zooming/focusing cameras. Obtained results of self-calibration techniques are not that reliable because many parameters must be estimated [Zha00].

All techniques are used to solve the following equations (3.14), (3.15) for all points in the field of view of the camera.

$$x_{img} = o_x + f_x \frac{r11X_w + r12Y_w + r13Z_w + T_x}{r31X_w + r32Y_w + r33Z_w + T_z} \tag{3.14}$$

$$y_{img} = o_y + f_y \frac{r21X_w + r22Y_w + r23Z_w + T_x}{r31X_w + r32Y_w + r33Z_w + T_z} \tag{3.15}$$

From the mathematical point of view, the calibration problem can again be classified into two classes.

- Linear techniques
  like [LT87] which are fast to compute but less accurate than nonlinear techniques.
- Nonlinear techniques
  Many techniques were published in the last decades like [Har94], [Gan84] and [Tsa86]. Some of them [Zha00], have even taken care of the radial distortions.

If the camera setup does not change, a calibration has to be done only once because the intrinsic and extrinsic parameters do not change. Exceptions are systems where the camera changes their parameters because of moving parts like a zoom, auto focus or also because of the heat created by the internal electronic components. The generation of a lookup table simplifies the handling of further images as long as the properties are not changed. This allows generation of calibrated images in linear time.

3.3 Stereo Camera Vision

A stereo vision system enables the estimation of the position of a point by triangulation using two cameras. Triangulation is, in fact, the easiest part. The bigger problem is to find the corresponding points for the triangulation. Such problems are already covered in many books and papers [TV98, BBH03] and it is named the correspondence problem. The discussion of such will start with the simple part, the stereo vision geometry and the triangulation.

3.3.1 Two Coplanar Pinhole Cameras

A simple stereo vision setup is to mount two monocular cameras with the same intrinsic parameters parallel to each other. Figure 3.5 shows the schematic of such a simple stereo vision system in 2D. A point P in the field of view of both cameras projects on to the left image plane the point p_l and on the right image the corresponding point p_r. The location of the point P can then be computed if the distance between the two centers of projection ($T = O_r - O_l$) is known as well as the *Effective Focal Length* f. Equations (3.16) and (3.17) describe the position of the point P. T is also known as *Baseline* and the difference between x_r and x_l is named *Disparity* $d = x_r - x_l$.

$$Z = f\frac{T}{d} \tag{3.16}$$

$$X = Z\frac{x_l}{f} \tag{3.17}$$

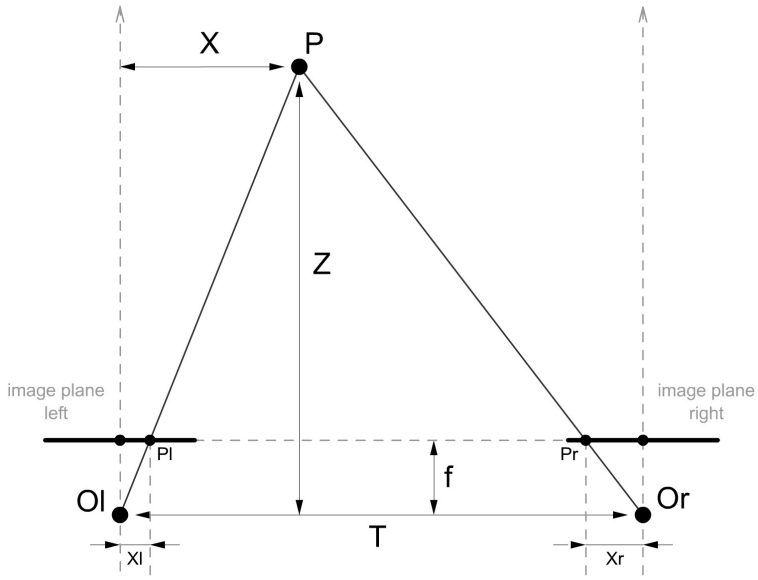


Figure 3.5: Schematic view of a stereo vision system with two coplanar pinhole cameras.

3.3.2 Epipolar Geometry

In this section, we want to describe how the search for a corresponding point can be minimized. First, we can say if we do not have any information about the geometry that the corresponding point p_r to a point p_l can be anywhere in the image. If we have a coplanar camera system like in Section 3.3 the two corresponding points must lie on the same vertical position on the image plane. In this way, the search can be restricted to a single line but most of the time we must handle systems where the cameras are not mounted coplanar. The solution for that problem can be found in the epipolar geometry shown in Figure 3.6. The epipolar geometry describes a plane which passes through the optical centers of the cameras, 0_l and 0_r, and the point P observed in the 3D world. The intersection of the plane π_p with the image planes creates two lines which are called *Epipolar Lines*. Therefore, the corresponding points for a point P in the image plane must lie on the epipolar lines. Hence, we can see that if the point P changes its position, the *Epipolar Plane* π_p rotates around the axis created by 0_l and 0_r and passes through the points e_l and e_r called epipoles which are always on the epipolar line. As a result of the epipolar geometry, we can say that even if the cameras are not coplanar, the search for a corresponding point can be restricted to a single line.

3.3.3 Essential and Fundamental Matrices

Figure 3.6 illustrates the Epipolar constrains. Now we will examine the mathematical size of the epipolar geometry. Our objective is to describe the epipolar line in a formula. Before we can start, we must introduce some new variables.

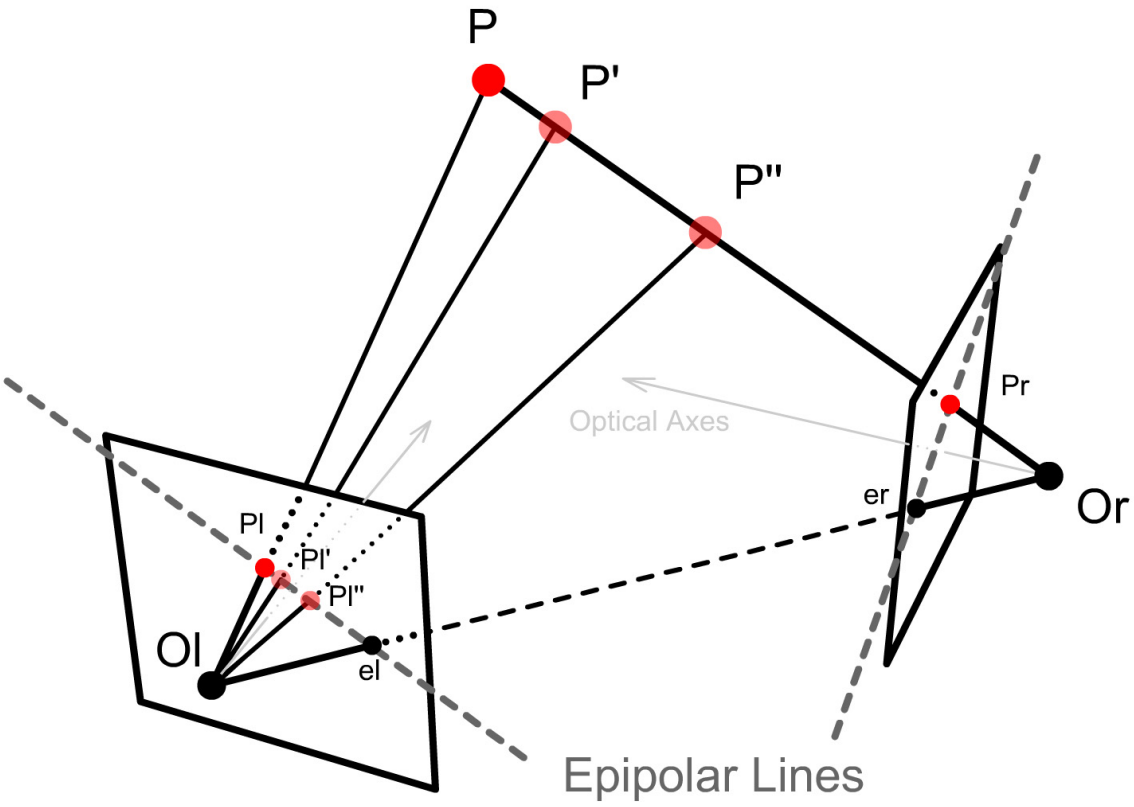


Figure 3.6: Epipolar geometry.

- $\vec{P_l} = \overrightarrow{O_l P}$ and $\vec{P_r} = \overrightarrow{O_r P}$
  Vectors to the point observed in the 3D World from the camera centers.
- $\vec{e_l} = \overrightarrow{O_l e_l}$ and $\vec{e_r} = \overrightarrow{O_r e_r}$
  Vectors to the epipoles from the camera center.
- $\vec{p_l} = \overrightarrow{O_l p_l}$ and $\vec{p_r} = \overrightarrow{O_r p_r}$
  Vectors to the point on the image plane corresponding to P.
- $\bar{p_l}$ and $\bar{p_r}$
  Corresponding points to P on the image plane in image coordinates.
- u_l and u_r
  Describes the lines along the epipolar lines.
- $\bar{u_l}$ and $\bar{u_r}$
  Corresponding lines to u_l and u_r but in image coordinates.

We assume that all camera properties like intrinsic and extrinsic matrices are known. The vector $\vec{P_l}$ can be transformed to $\vec{P_r}$ by a rotation R and translation $\vec{T}$ (3.19). Therefore the cross product must be zero (3.19). This constraint is also known as the *Coplanarity*

Constraint. The cross product in a normal $\mathbf{R}^3$ can also be written as pure matrix multiplication (3.20). We can now simplify (3.19) to (3.23) by replacing the cross product (3.21) and defining E (3.22). E is the *Essential Matrix*. Therefore, we can use (3.1) and (3.2) to calculate the vectors to the corresponding points on the image plane (3.24) and also identify the intersection with the image plane u_r (3.25).

$$P_l = Z\frac{x_l}{f} \tag{3.18}$$

$$(R^\top \vec{P}_r)^\top \vec{T} \times \vec{P}_l = 0 \tag{3.19}$$

$$\vec{T} \times \vec{P}_l = S\vec{P}_l = \begin{pmatrix} 0 & -T_z & T_y \\ T_z & 0 & -T_x \\ -T_y & Z_x & 0 \end{pmatrix} \begin{pmatrix} P_x \\ P_y \\ P_z \end{pmatrix} \tag{3.20}$$

$$(R^\top \vec{P}_r)^\top S\vec{P}_l = 0 \tag{3.21}$$

$$E = RS \tag{3.22}$$

$$\vec{P}_l{}^\top E\vec{P}_l = 0 \tag{3.23}$$

$$\vec{p_l}{}^\top E\vec{p_l} = 0 \tag{3.24}$$

$$\vec{u_r}{}^\top E\vec{p_l} \tag{3.25}$$

The *Fundamental Matrix*, like the essential matrix, enables us to recover the full epipolar geometry. It can be recovered using information of corresponding points in the images. No knowledge about the extrinsic and intrinsic parameters is necessary. One famous algorithm used to perform this is the *Eight-point Algorithm* [Har97] which solves the problem based on a linear equation via a *Single Value Decompression* (SVD). Unlike the essential matrix he fundamental matrix is based on *pixel coordinates* and not on camera coordinates. The relation between the fundamental and the essential matrix is given by (3.26).

$$F = Mint_r^{-T} EMint_l^{-1} \tag{3.26}$$

The other properties of the fundamental matrix are the same with the restriction that the system is based on the pixel coordinates (3.27) and (3.28).

$$\bar{p_l}^\top F \bar{p_l} = 0 \tag{3.27}$$

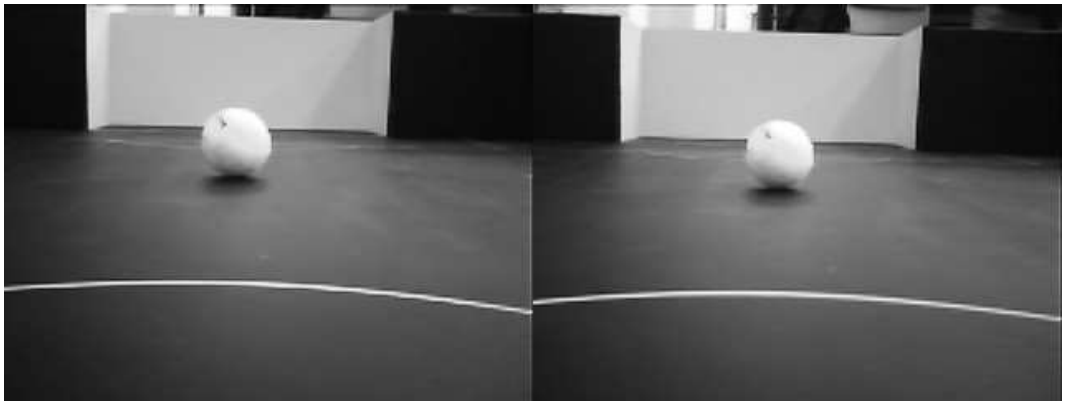
$$\bar{u_r} = F \bar{p_l} \tag{3.28}$$

3.4 Rectification

The rectification of a stereo image pair is the process of transformation on the images in such a way that the pairs of conjugate epipolar lines become collinear and parallel to one of the image axes, usually the horizontal one [TV98]. A rectified image pair therefore has the same geometrical properties as an image pair of a coplanar pinhole stereo system.
After we find such a transformation, this transformation can be combined with the camera calibration to create a lookup table which calibrates and rectifies an image at once. This lowers the complexity of finding correspondences because we only need to search only in a horizontal line instant on a computed line u_r which is different for every point p_l (3.28). Figure 3.7 shows an unrectified and rectified image pair.

3.5 Summary

This chapter showed the basic geometric constraints of monocular and stereo camera systems: the relation of a point in the image with the corresponding point in the 3D world and how this relation is described by the intrinsic matrix and extrinsic matrix. The epipolar geometry with the essential matrix and the fundamental matrix is covered in the Section 3.3.3. Finally, the rectification was discussed regarding how it can support the search for correspondences in stereo images.



(a) Not rectified image pair

(b) Rectified image pair

Figure 3.7: This figure shows two image pairs where the lower one is rectified. Correspondences must be found in the image (a) on the epipolar line going cross the image while in the lower pair (b) the epipolar lines are horizontal and the camera distortions are removed.

Chapter 4

3D Vision

The biggest eye can never see as much as two eyes.

(A saying)

There are different ways how 3D information can be drawn from monocular and stereo vision systems. Depth information can be computed in monocular systems from shading, texture, motion and from geometric knowledge of the environment. This chapter discusses the background for the techniques used in the implementation. Shape from shading or texture are not discussed in this master thesis but further reading can be found in [TV98]. The first part will focus on monocular systems while the second part discusses creating a depth map based on the stereo image information, and finally a summary concludes this chapter.

4.1 Geometric Knowledge of the Environment

If the intrinsic parameter of the camera system and the objects in the surrounding area are known, the distance to an object can be estimated only by its size or position in the image.
An *object model* represents the known properties of an object in the world. Such properties can be the color, size and the relative position to the ground. The size is defined as a radius or a diameter in the case of a sphere and the width, height and depth are used to describe a box. Another important parameter is the anchor point or the origin from which the object is described. This anchor point is important because when we are speaking of the position of the object, we refer to the anchor point of the object. The anchor for a spheric object is defined by its center. We will place the anchor point for a rectangle and boxes in their object centers.

4.1.1 Known Object Size

Spheres are especially suitable to determinate their distance by their detected size in the image [SRTSB92], because a spherical object projects a circular arc on the image plane which is easy to detect (Section 2.2.2). The object size in the image is directly related to the distance from the object to the camera (4.1). d represents the distance from the object to the camera, f the focal length, s_{obj} the measured size of the object in the image and s_{model} the known object model size. Figure 4.1 shows the projection of an object on the image plane. We are interested in the vector r_{world}. This vector can be computed after d is solved using equations (4.2) and (4.3).

$$\frac{s_{obj}}{f} = \frac{s_{model}}{d} \tag{4.1}$$

$$r_{img} = \sqrt{f^2 + x_{img}^2} \tag{4.2}$$

$$\frac{r_{img}}{f} = \frac{r_{world}}{d} \tag{4.3}$$

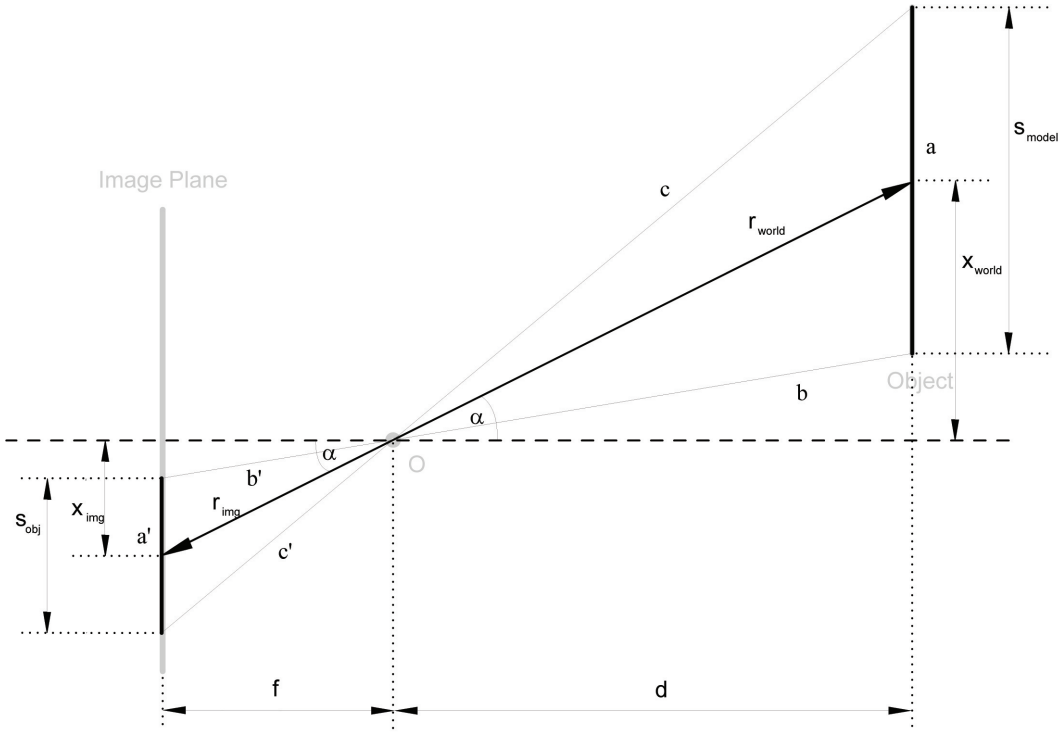


Figure 4.1: Shows the relation between object size and the projected size of the object.

We assumed that in this section a sphere projects a circle onto the image plane. This is

only the case if the sphere lies on the optical axes of the camera. Otherwise, it will project an ellipsoid on the image plane. But the aberrance for a circle in our case is minimal so that we can assume that a sphere always projects a circle onto the image plane. Details and solutions to the sphere's localization estimation can be found in [SRTSB92].

4.1.2 Known Relative Position to the Floor

If the camera angle α_{camera} to the floor as well as the mounted height of the camera h are known, the distance d to an object can be computed if (a) the object lies on the floor or (b) the relative position of the object h_r to the floor is known. Figure 4.2 shows how such a situation appears. If the projection of the point P with $< x_p, y_p >$ is detected in the

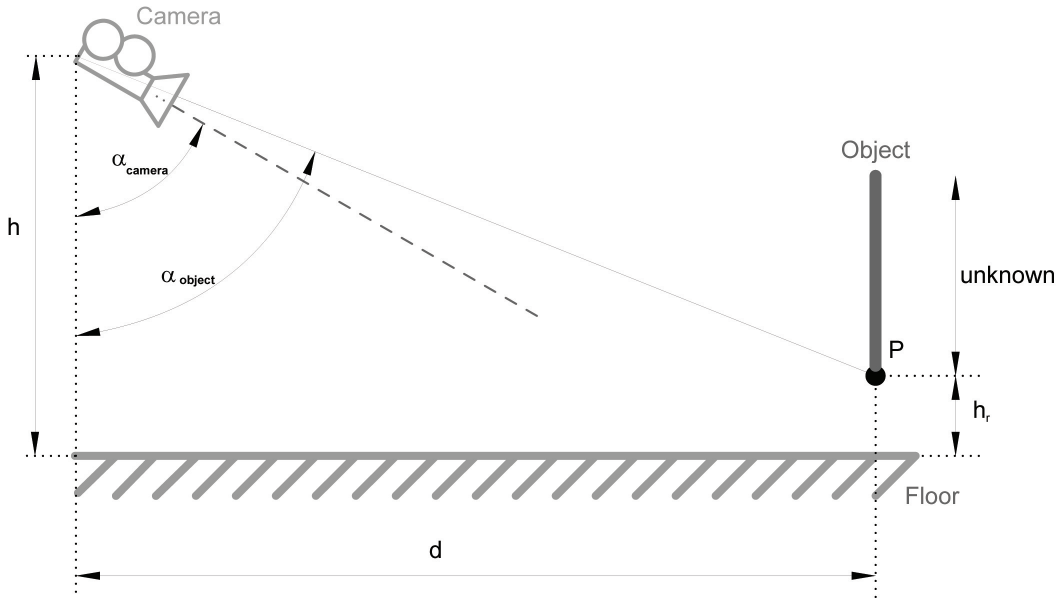


Figure 4.2: Show the triangulation of an object if the environment is known and the object size not.

image, the angle α_{object} can be computed using equation (4.4) and hence the distance d to the object by equation (4.5)

$$\alpha_{opject} = \arctan\left(\frac{y_p}{f}\right) + \alpha_{camera} \tag{4.4}$$

$$d = \frac{h - h_r}{\tan \alpha_{opject}} \tag{4.5}$$

4.2 Stereo Vision

Estimating the distance of a certain point seen in both images of a stereo system was already discussed in Section 3.3. It was also mentioned that the bigger problem lies in determination of the correspondences between two pixels.

4.2.1 Correspondences

[BBH03] published an overview over techniques to solve the correspondence problem. He divided the techniques into two groups: the local and the global methods. *Local Methods* are determine correspondences by observing information in the neighborhood of the pixel that is involved in the current search. *Global Methods* use the underlying information from scan lines (epipolar lines) or the entire image to find correspondences. Figure 4.3 shows in the upper figures a scan line going horizontal through the images and in the lower figures a graph which represents the intensity value of the gray image under the scan line. The bright shadows next to the curves in the lower figure show the intensity of the scan line from the opposite scan line.
Brown describes in the following citation the benefits and problems with global and local properties when solving the correspondence problem.

> *Local methods can be very efficient, but they are sensitive to local, ambiguous regions in images. Global methods can be less sensitive to these problems since global constraints provide additional support for regions difficult to match locally. However, these methods are more computationally expensive.*
>
> [BBH03] about local and global methods

Since global methods take more effort to compute they are not used in the implementation and are not further discussed. Local techniques can be classified into two more groups: the *Correlation-Based Methods* and the *Feature-Based Methods*.

Correlation-Based Methods

The idea of a correlation-based method is to find for a small image patch a corresponding patch in the other image. The correlation-based methods are also called *Area-Based Methods*. There are functions known to compare two areas for similarity or sameness. Hirschmüller compared in his work in 2001 [Hir01] different standard correlation methods in terms of real-time capability on standard PC hardware. Let I_1 and I_2 represent intensities in two windows. u, v and d are described in Figure 4.2.1.

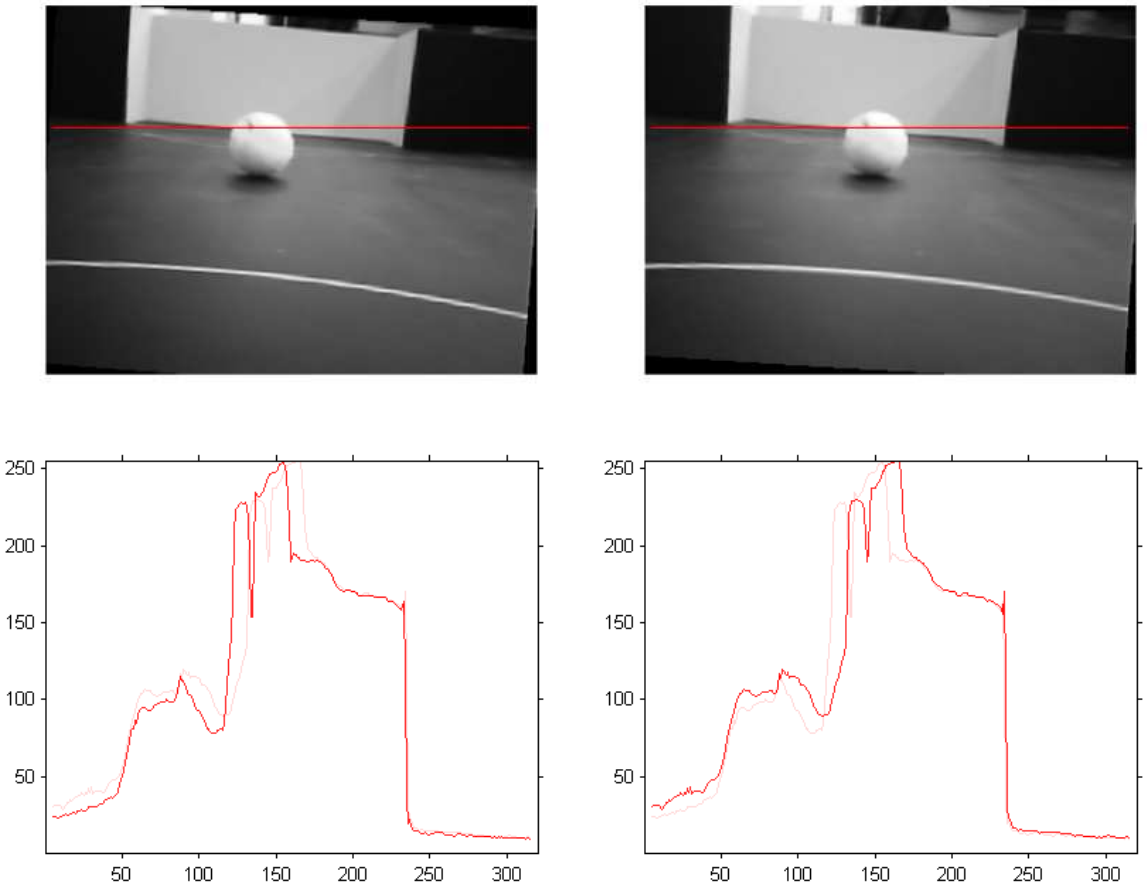


Figure 4.3: The upper image pair shows a scan line through two rectified images. The lower ones show the intensity value of the underlaying pixels of the scan line.

- *Normalized Cross Correlation (NCC)* normalizes the mean and variance to make the correlation insensitive to radiometric gain and bias (4.6).

$$NCC = \frac{\sum_{u,v} I_1(u,v) - \bar{I}_1) \cdot (I_2(u+d,v) - \bar{I}_2))}{\sqrt{\sum_{u,v}(I_1(u,v) - \bar{I}_2)^2 \cdot (I_2(u+d,v) - \bar{I}_2)^2)}} \tag{4.6}$$

- *Sum of Squared Differences (SSD)* is simpler to compute but not normalized and therefore sensitive to distortions (4.7).

$$SSD = \sum_{u,v}(I_1(u,v) - I_2(u+d,v))^2 \tag{4.7}$$

- *Normalized SSD* is an extended version of SSD with a normalization (4.8).

$$\text{Normalized } SSD = \sum_{u,v}\left(\frac{(I_1(u,v) - \bar{I}_1)}{\sum\sqrt{(I_1(u,v) - \bar{I}_1)^2}} - \frac{(I_2(u,v) - \bar{I}_2)}{\sum\sqrt{(I_2(u,v) - \bar{I}_2)^2}}\right)^2 \tag{4.8}$$

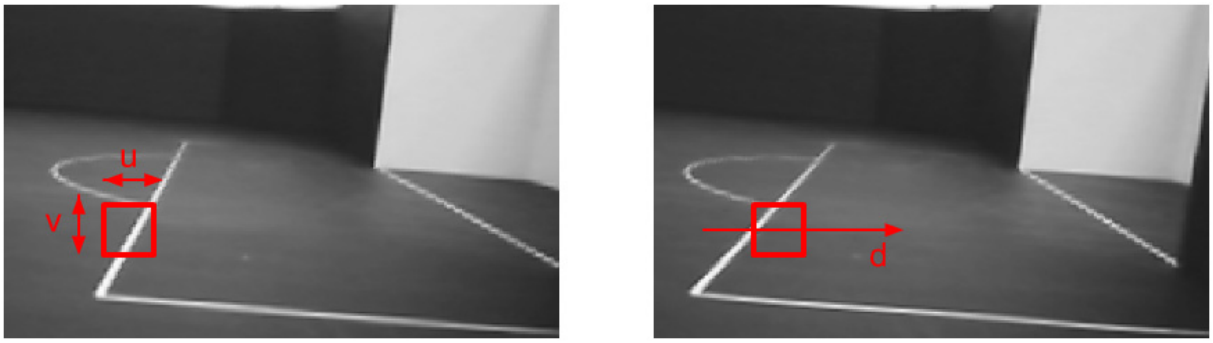


Figure 4.4: Correlation-Based method. An area will be compared with areas on the epipolar line in the stereo-related image.

- *Sum of Absolute Differences (SAD)* represents a modified version of SSD where no square function is involved to simplify the computation (4.9).

$$SAD = \sum_{u,v} |I_1(u,v) - I_2(u+d,v)| \tag{4.9}$$

- *Rank*. The rank represents the order of the center pixel of the image patch $\hat{I}_{u,v}$ in an ordered list with its neighbors (4.10). Figure 4.2.1 shows a sample of rank computation.

$$\begin{aligned} Rank = \quad & \textstyle\sum_{u,v} \hat{I}_1(u,v) - \hat{I}_2(u+d,v) \\ & \hat{I}_k(u,v) = \textstyle\sum_{m,n} I_k(m,n) < I_k(u,v) \end{aligned} \tag{4.10}$$

$$\begin{array}{cccc} 255 & 40 & 150 & \\ 200 & 162 & 10 & \rightarrow 5 \\ 202 & 140 & 20 & \end{array}$$

Figure 4.5: Example of a rank value computation

- *Census*. The Census value is a bit-string (8Bit). Every bit represents a connected neighbor. If one neighbor is bigger than the center the related bit is given the value of one otherwise zero (4.11). Figure 4.2.1 shows a sample of rank computation.

$$\begin{aligned} Census = \quad & \textstyle\sum_{u,v} Hamming - Distance(\check{I}_1(u,v) - \check{I}_2(u+d,v)) \\ & \check{I}_k(u,v) = Bit - String_{m,n}(I_k(m,n) < I_k(u,v)) \end{aligned} \tag{4.11}$$

The rank and census techniques are insensitive to brightness changes between the images due to their method of computation. Both rank and census are used on FPGA implementations because of their simple computation. [BN98] introduced “Ordinal Measures for

$$\begin{array}{ccc} 255 & 40 & 150 \\ 200 & 162 & 10 \\ 202 & 140 & 20 \end{array} \rightarrow 01001011$$

Figure 4.6: Example of a census value computation.

Image Correspondence" which takes care of internal structures of compared area patches to minimize correspondence mistakes. The next section will focus on feature-based implementations.

Feature-Based Methods

Unlike the correlation-based methods, the feature-based methods are searching corresponding features and not corresponding search windows in the other image. The image features can be edges, lines or other higher level objects like image segments. [DA89] compares various stereo methods both correlation-based and feature-based, while the feature-based methods are focused on line features. [BT99] presents an algorithm based on extracted image segments to compute slanted surfaces. Similar to [DA89], line segments were used to estimate the lines, 3D orientations and positions in the master thesis [Ent05].
The benefit of a feature-based stereo vision algorithm lies in the smaller amount of data which must be compared. And the image features can be directly used for further analysis like finding shapes or structures in the image. Because of this feature-based methods are faster than correlation-based approaches but at the cost of an incomplete depth map. The depth information can only be computed on image features and must be interpolated for the space in-between if necessary as in [BT99].

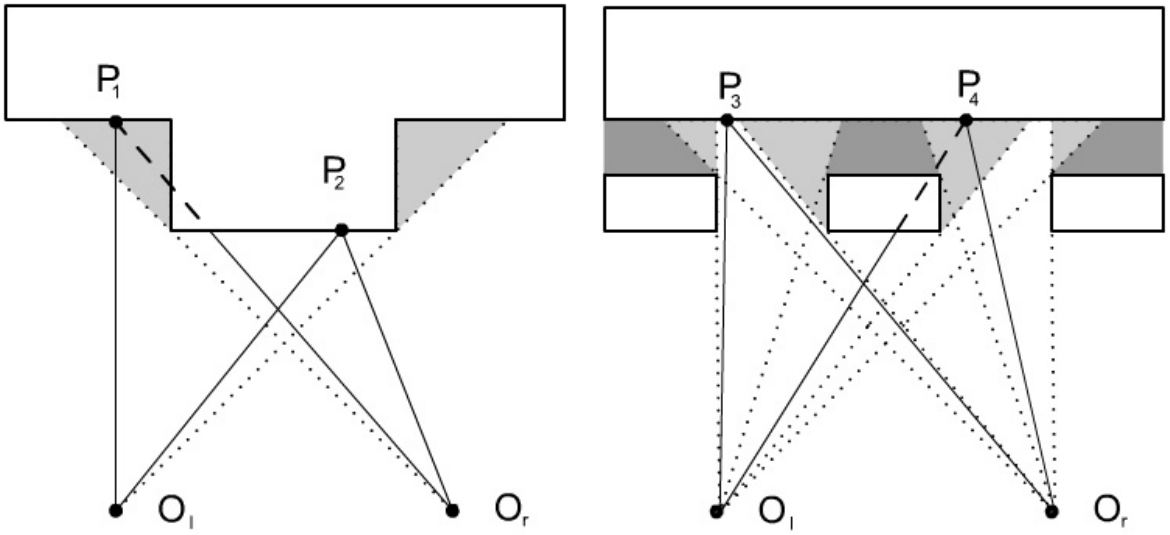
4.2.2 Occlusion

Dhond divided occlusions in his paper into two groups by using the following definition.

> *An occluding object is classified as narrow if its pixel-width is narrower than (a) the local support neighborhood used in the spatial hierarchy for global matching, or (b) the largest disparity difference between the background and foreground scene objects.*
>
> [DA92] definition of narrow occluding objects

The first group (a) of the occlusions defined is shown in Figure 4.7.a. They occur if the observer views a scene with a 3D structure on it. The second group (b) is presented in Figure 4.7.b. They occur if the observer looks through a structure onto the scene. Such a structure can be a fence or a window which creates a scene with background and foreground objects.

As long as there are no transparent or occluded areas, every image feature can only be matched to one corresponding feature in the other image. Disparity outliers in the local neighborhood or disparity discontinuities can be used to detect occlusions, because in natural images, the disparity varies nearly smoothly throughout.



(a) (b)

Figure 4.7: Two narrow occluded points P_1 and P_4, while P_1 is the most common one. The light gray areas mark half-occluded areas, while the dark gray areas are not visible to both cameras.

Techniques to detect, reduce and model occlusions are known and listed in [BBH03]. The most common ones are listed here.

- Left-Right Matching
  Correspondences are determined from the left to the right side and vice versa. The differences between the two results are handled as occlusions.
- Depth Map Discontinuities
  A discontinuity in the depth map indicates occlusions.
- Multiple Cameras
  An additional view reduces the occlusions but increases the complexity

4.3 Summary

This chapter discussed different ways how to extract 3D information out of images. The first section presented techniques for monocular systems. Only the approaches used in the implementation like depth estimation in a known environment where discussed. The

second part of this chapter focused on stereo vision and the different techniques to extract depth information based on two views of one scene. Related correlation techniques like global and local methods were also described. The local method were discussed in more details because the feature-stereo algorithm used in the implementation falls into this group. The stereo vision section ends with the description of known problems based on occlusions. Thus far, all fundamental methods for understanding the implementation have been discussed. The next chapter will describe the actual implementation.

Chapter 5

Implementation

"Let´s start with the three fundamental Rules of Robotics - the three rules that are built most deeply into a robot's positronic brain." In the darkness, his gloved fingers ticked off each point.
We have: one, robot may not injure a human being, or through inaction, allow a human being to come to harm."
"Right!"
"Two," continued Powell, "a robot must obey the orders given it by human beings except where such orders would conflict with the First Law."
"Right!"
"And three, a robot must protect its own existence as long as such protection does not conflict with the First or Second Laws."

(I. Asimov, "Runaround" - Powell and Donavan discussing the laws for robotics)

In previous chapters, we talked about methods and techniques to implement a vision sensor. The topic of this chapter is the implementation of the vision sensor together with its hardware. The first part describes the hardware used and the second part the implemented software.

5.1 Hardware

This section covers hardware used by describing the Tinyphoon robot, its vision module and finally the programming environment.

5.1.1 The Tinyphoon Robot

The Tinyphoon robot was designed as an autonomous robot. The size and also the driving concept is similar to the robots used in the expected AMiroSot league of the FIRA organization.

Mechanics: The chassis of the Tinyphoon is made of fiber-reinforced plastics and houses two DC motors and a rechargeable battery. It is designed to have a very low-lying center of gravity. The motors can speed up the robot to more than $2.5m/s$ with an acceleration of more than $5m/s^2$. The two high precision encoders on the motors function with a resolution of 512 steps. This reduces the tolerance to $5mm/m$.

Motion Unit: The motion unit is shown on the left side in Figure 5.1. It controls the motors and is equipped with various sensors: two two-axis acceleration sensors, a gyro sensor and a magnetic field sensor. A processor from Infineon's [1] XC series is used for collecting data from the analog sensors and for recording the movement of the wheels. The motion control is implemented using an Analog's Blackfin BF533 processor. The unit is connected via radio to a PC or other robots and with a serial communication to the *Strategy - WMR (World Model Repository) - Reasoning Unit*

Strategy - WMR (World Model Repository) - Reasoning: This is the *brain* of the robot, decisions what the robot should do next based on the sensor inputs and the history of the last actions are made here. This unit is connected via serial communication to the motion and with a *Serial Peripheral Interface Bus* (SPI) to the vision unit. The purpose of the WMR is to provide a general interface to the sensor and to create sensor fusion to verify received information. Currently strategy, WMR and reasoning are located on one core module. In the next version, this unit will be split into to physically separate hardware modules.

Vision Unit: The vision unit augments the Tinyphoon robot with stereo vision capability. Two CMOS cameras with a resolution of up to 640 x 480 pixels are connected via *PPI (Parallel Peripheral Interface)* to a dual core DSP. Figure 5.1 in the left image shows the vision hardware. The unit recognizes structures related to known objects in the environment. This data is submitted via SPI to the WMR.

Communication: Different strategies for communication are possible. All hardware layers support CAN and serial communication. The connection to systems like PCs or other robots are possible via a serial wireless communication. The next version will also support time-triggered protocols. More details on the communication on mobile robots can be found in [Kry06].

[1] www.infineon.com

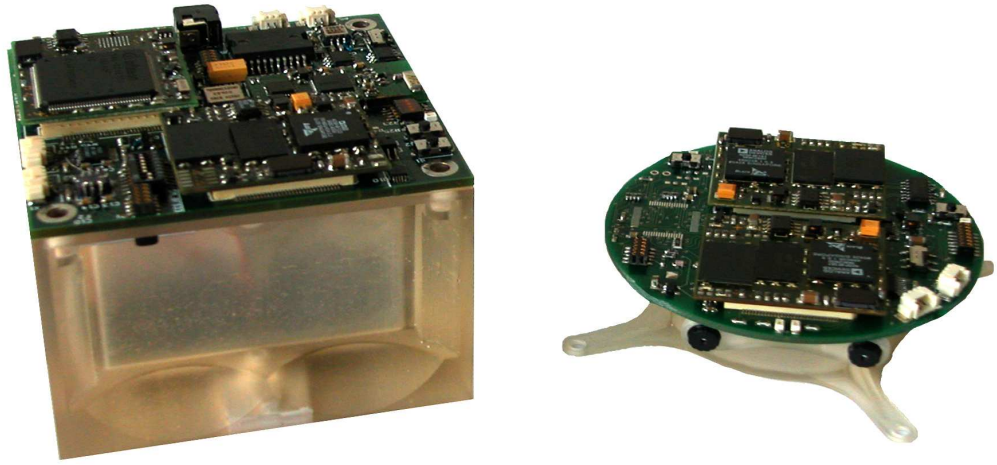


Figure 5.1: Left motion module and right, the vision module.

5.1.2 The Vision Module

The vision module on the Tinyphoon robot is equipped with two Blackfin DSP processors. One of them is a dual core DSP. The dual core DSP is used for image processing. The second Blackfin hosts the Strategy - WMR - Reasoning unit. Images are taken by two CCD CMOS cameras[Omn05] and transported via PPI directly into the SDRAM of the processor. The camera properties like the shutter-speed can be manually defined during the camera initialization. This is important because of the black background in the test environment where cameras in auto-mode tend to overexpose.
The mechanical module of the vision system is mounted on a turnable platform to rotate the system while the robots stays still. The strategy defines the mode for the vision sensor. The mode defines the thresholds used in the detection. Those thresholds are the camera settings like shutter-speed, gain[2] and camera resolution, as well as the setting defining the thresholds for the detection. The strategy also submits models of the objects to detect and is returned the recognized objects. Both mode and models are described in Section 5.2.6

5.1.3 Dual-Core DSP and VisualDSP

The image processing is done on an Analog Devices dual-core DSP [Ana05] mounted on a core module CM-BF561 [Blu06] from Bluetechnix[3]. The programming environment VisualDSP extends the normal ANSI C with processor-specific function calls. Such calls are then encapsulated in assembler statements and allowed to add or subtract two or in special

[2]The gain setting can be viewed as the camera's sensitivity.
[3]www.bluetechnix.com

cases even four registers at once in one processor cycle.
The memory management is controlled by the *Linker Description File* *.ldf. This file controls where functions and variables are mapped in the memory . This is important in a dual-core system where, in our case, a three-level-memory must be coordinated and partially shared between two cores. Figure 5.2 shows the memory structure of the core module. We can see in this figure the two L1 memory section where one is designated for each core, the shared internal memory L2 and the external L3 memory named as SDRAM. The access speed to the SDRAM is limited by the BUS clock rate which is up to ten times slower than the internal clock rate [Ana05]. The L1 and L2 memories are placed internally and can be accessed at the processor clock speed. L2 and L3 memory can be addressed by both cores and can be used to synchronize the cores and/or to exchange data between them.
Time-relevant program sections are placed in the L1 memory where they can be accessed at the maximum clock rate. A problem occurs if both cores are trying to write data into the same memory location in the shared memory. This problem is solved by using shared variables to synchronize the parallel running programmes if they are writing into shared memory sections.

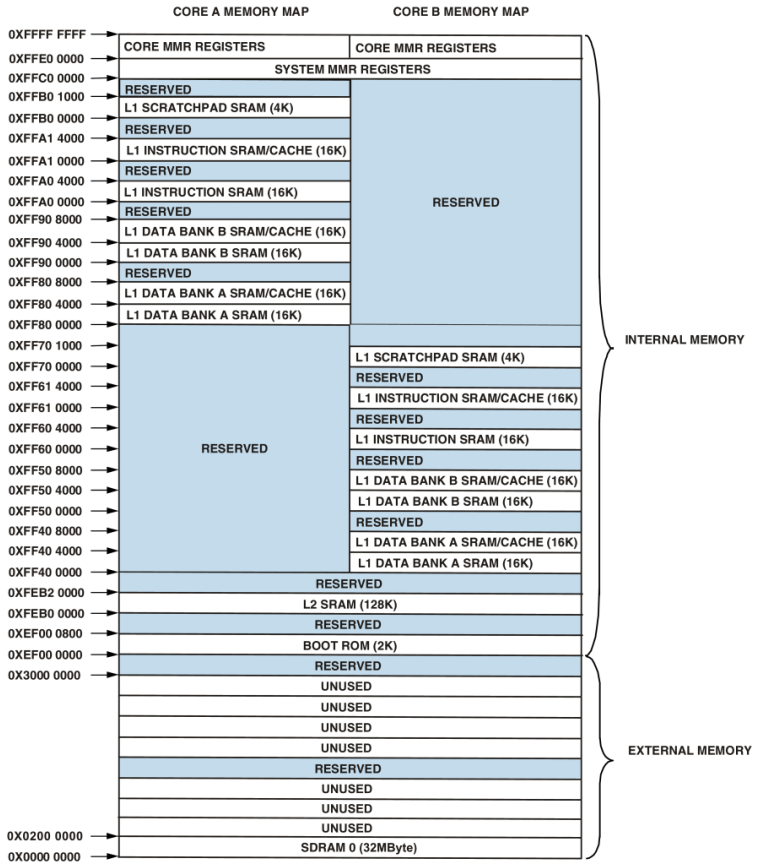


Figure 5.2: Memory structure of the CM-BF561 Core Module.

5.1.4 Summary

This section provided the reader with general information about the robot. The subsection about the vision module described that the operating mode of the vision system and the models of objects to detect are defined by the connected strategy unit. Techniques to use and manage a dual core system were also discussed as well as the programming environment VisualDSP. The last subsection described how a dual core system synchronizes data between the cores and the related problems.

5.2 Software

This section covers how the strategies were implemented on the system used and which variations were made on existing methods to run them in real-time on the DSP. The segmentation algorithm as well as the edge, blob and line detection were implemented as a scan-line algorithm which moves only once from the top left to the bottom right, line by line over the image data. The DMA-Controller was used to implement a ring buffer which holds only data necessary for processing in the processor fast accessible L1 memory. Every time the algorithm processes a line, the DMA-Controller moves the results of the last line into the SDRAM and the next line to process into the ring-buffer in the L1 cache memory.

5.2.1 Edge Detection

The edge detection is based on the canny edge detector. This detector has five internal steps listed in Section 2.1.4. The first four of them are implemented as shown in Figure 5.3. The DMA-Controller copies an image line into the L1 memory where it will be first reduced to a gray-scaled image by taking only the Y (chroma) information of the YUV image. After each processed line, all ring buffers rotate. While the color reduction, Gaussian, Sobel and non-maxsuppression are in process the last finished line of the non-maxsuppression will be copied back to the SDRAM and the next one to process will be copied into the first ring buffer. This is possible because the DMA-Controller can work without the *ALU (Arithmetic Logic Unit)* after the controller is initialized and started [PH98]. The fifth step, the hysteresis threshold, is realized by using a variation of the Rosenfeld segmentation algorithm of Section 2.3. A segment is in an edge image equal to a group of connected edges. If we remove all edge groups with a gradient value under the threshold τ_h, we have the same effect as at the hysteresis threshold.
The final edges after the hysteresis threshold are stored in a structure shown in Listing 5.1. This structure allows us to hold all edges in the L1 Memory but at the cost of slow access-speed to random chosen edges. On the other hand this structure enables rapid access to edges sequentially in a scan-line algorithm which is important for the following line detection.

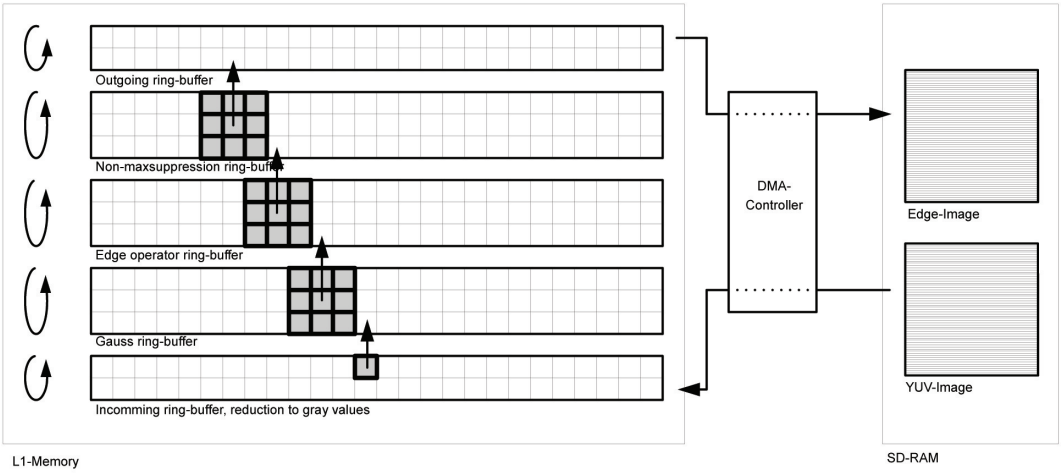


Figure 5.3: Memory management on the edge detection by using the DMA-Controller.

```
typedef struct {
    unsigned char iGradient;              // Edge gradient
    unsigned char iGradientDirection;     // Edge direction with values between 0x00 and 0xFF
    unsigned short x;                     // X position of the edge
} t_EdgeDataElement;

typedef struct {
    unsigned long iEdgeIndex;             //Index to the first edge of the row in ptEdgeData
    unsigned short iEdgesInRow;           //Number of edged in the row
} t_EdgeDataIndex;

typedef struct {
    unsigned short iWidth;                //Image width
    unsigned short iHeight;               //Image height
    t_EdgeDataElement *ptEdgeData;        //Pointer to the first edge
    unsigned short iSizeEdgeData;         //Avaliabe memory size at ptEdgeData
    unsigned short iCountEdges;           //Number of edges stored at ptEdgeData
    t_EdgeDataIndex *ptIndex;             //Pointer to the array t_EdgeDataIndex[iHeight]
} t_EdgeHdl;
```

Listing 5.1: Structure to store an edge image.

5.2.2 Line Detection

The line detection algorithm implemented is based on local and global properties, and works in three steps.

1. **Line Segment Detection:**
   This step only detects lines which are based on connected edges shown in Figure 5.4.b. It searches for connected edges similarly to how the Rosenfeld algorithm does in Section 2.3, when seeking connected components but with the addition that an edge will only be added to a group if it corresponds to a certain threshold with the average gradient direction. Instead of the lookup table a table of line structures is used which stores the first and the last occurrence of an edge in the segment as a line

starting-point or ending-point.
It is important to use only the first edges to build an average gradient direction value for a threshold to avoid the detection of flat arc-looking curves. The first occurrence of an edge on a line is stored as the line start-point. Every time the scan-line algorithm passes an edge, it will check if it lies next to a line end-point and if the average gradient direction is in the angle range threshold. In this case, it will then be stored as a new end-point of the line. A special case occurs with lines which become flat from the right to the left side because of the scan from the left to the right side. This case can be detected by their average gradient direction. In such cases We have to compare the edge location with the lines start-point and not with the lines end-point, and if it matches it has to be stored as a new start-point. The threshold for these detection steps are:

- *Angle range threshold* to indicate if an edge matches the current line segment.
- *Line min length* to remove lines which have an edge-count under this limit.

Before we store the detected segments, we place the line segments start and end points in a way that all lines have the edge gradient direction looking to a chosen side relative to the line direction.

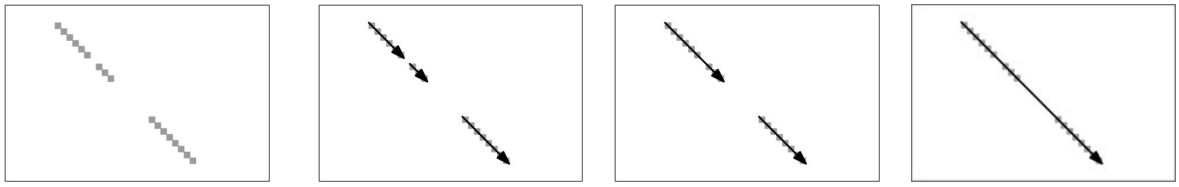
2. **Local Line Segment Connector:**
The local line connector connects line segments from the previous step if the start and endpoint are in a certain pixel range as well as line direction. This step is shown in Figure 5.4.c. Used thresholds are:

   - *Angle range threshold* to indicate if two lines are matching.
   - *Line min gap* is the pixel range in which the algorithm searches for a corresponding end point next to a line start point.

3. **Global Line Segment Connector:**
Global Muff line properties are used to find line segments on the same line to connect. This is presented in Figure 5.4.d. The Muff line properties represent where a virtual line in the direction of the detected line segments of the previous step would cross the border of the image. The Muff space was used to distinguish between line segments in different directions. The Muff line properties were discussed in Section 2.2.1. Line segments are therefore connected by satisfying these thresholds:

   - *Global property range threshold:* Range of the maximum allowed differences in the Muff space.
   - *Line max gap:* Represents the maximum distance which is allowed between two line segments to connect them.

The probability that a detected line really exists can be computed by the relation of the detected line length and the underlaying pixels.



(a) Source edge image (b) After step 1 (d) After step 2 (c) After step 3

Figure 5.4: Implemented line detection in different processing stages

5.2.3 Image Segmentation and Blob Detection

The camera provides a UYVY 4:2:2[4] color image I_{yuv}. This image is reduced to 28 colors plus white and black and represented by 32 bits per pixel I_{i32} (5.4). This representation simplifies the comparison between colors. The edge detection is done in the same function like the color reduction to avoid two equal accesses to the input image. Most of the computing power of the controller is used by the edge detection. Equations (5.1) - (5.4) describe the color transformation of the input image. Y represents the luma, V and U the chroma information of a pixel in YUV format. Because of the thresholds in (5.2), the reduced color depends more strongly on the chroma as on the luma information.

$$b = \mathrm{Y} - \sqrt{((V-128)^2 + (U-128)^2)} \tag{5.1}$$

$$c = \begin{cases} 1, & \text{if } b < \epsilon_{black} \\ 2, & \text{if } b > \epsilon_{white} \\ \frac{arctan2i(u,v)}{10} + 3, & \text{else} \end{cases} \tag{5.2}$$

$$p = \mathrm{0x00000001h} << c \tag{5.3}$$

$$I_{i32} = |_{x=1}^{x=width-1} |_{y=1}^{y=height-1} p(I_{yuv}(x,y)) \tag{5.4}$$

I_{c32} is a noise-reduced version of the image I_{i32}. An opening operation as the one described in Section 2.3.2 with a bitwise erosion followed by a bitwise dilation was used to generate I_{c32} from I_{i32}. A pixel in I_{c32} is now a 32-bit value where each bit represents a color and the zero bit stands for no color information. Blobs are generated by using the Rosenfeld segmentation algorithm of Section 2.3 and the colors of interest defined by the object models from the image I_{c32}. These blobs are then candidates for detected objects as you can see in Figure 5.7.

[4]Y sampled at every pixel, U and V sampled at every second pixel horizontally on each line.

5.2.4 Circle Detection

The circle detection is only applied on the area in and around a blob related to a spherical object model. The algorithm used is similar to the Hough Transform for Circles described in Section 2.2.2. But a line L_i is only drawn[5] for an edge in the blob with predefined borders toward the gravity center of the blob color *blob.cc*[6]. The circle center will then be represented as a peak where all the lines are intersecting. Equation (5.5) shows the line equation. We assume that *blob.cc* will be near the real circle center C_c. Hence it is only necessary to draw the lines in a small patch around *blob.cc*. The following two filters are used to skip edges which do not belong to the circle border.

$$L_i = x_i + tan(\alpha_i) \times y_i \qquad \alpha_i \text{ ... gradient direction} \tag{5.5}$$

An edge will be skipped if

- Filter 1:
  The edge is overlaid by a color entry
  $I_{s32}(edge.x, edge.y) \oplus S_{c32} > 0$
  I_{s32} is a bitwise erosion on I_{c32}
  $\oplus$ .. bitwise AND
  S_{c32} represents the color of the shape in 32 bit format
- Filter 2:
  The edge gradient does not look toward the gravity center of the color
  $|tan(\alpha_i) - \arctan(x_i - x_b / y_i - y_b)| < \epsilon_{angel}$
  α_i ... gradient direction

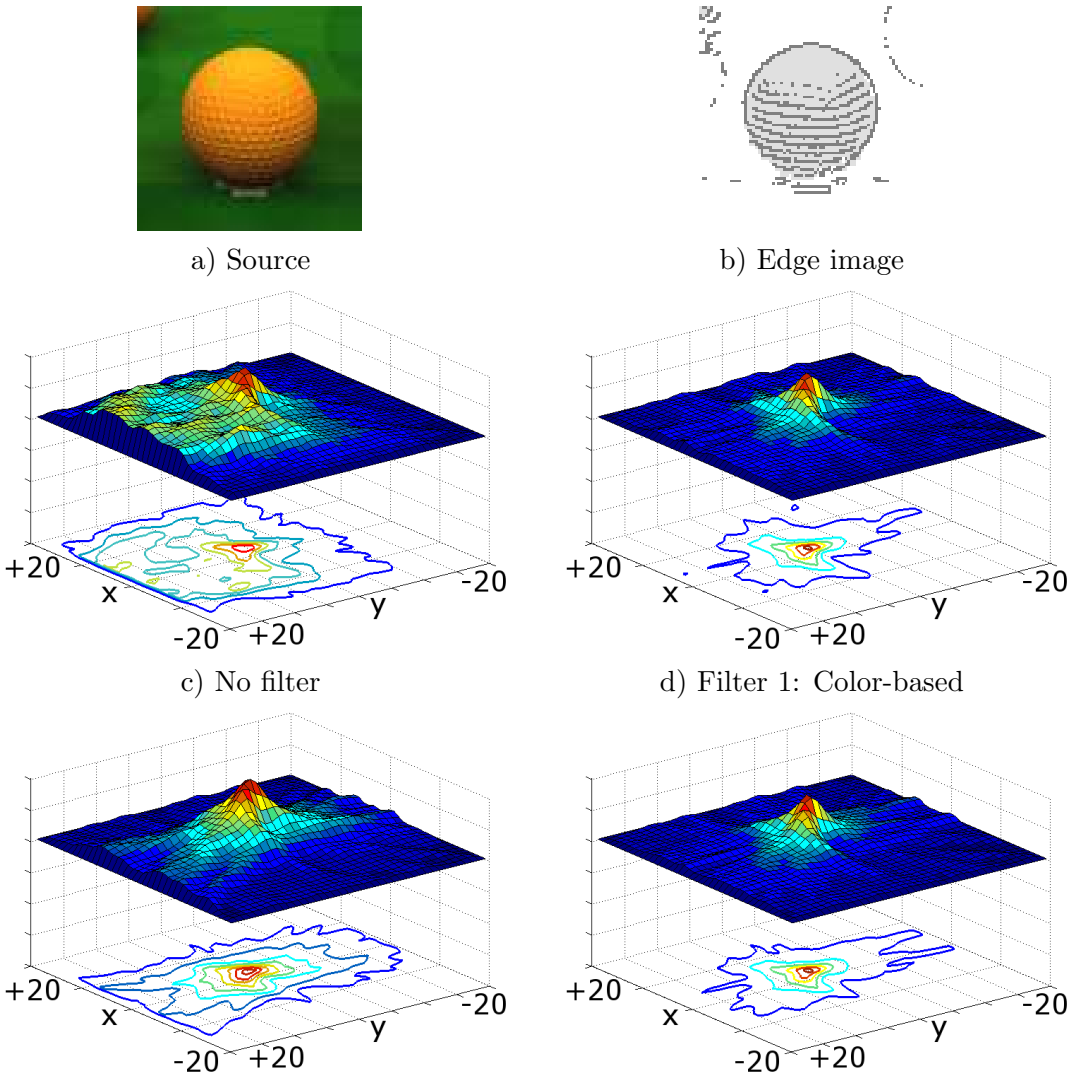
Filter 1 will remove edges generated by fine structures on the ball which appear on spheres like golf balls. Golf balls are used in the MiroSot league. The graphs in Figure 5.5 show how the filters changed the form of the peak in the Hough space corresponding to the sphere shown in (a) with its edge image (b). We can see that the peak which represents the circle center becomes clearer visibly by applying the represented filter.

The shape center test T_c is used to judge the detection. Equation (5.6) shows when the T_c fails.

$$T_c = \begin{cases} \text{failed}, & \text{if } \Sigma\ Edges < \epsilon_{MinEdges} \\ \text{failed}, & \text{if } \frac{PeakSize}{\Sigma\ Edges} < \epsilon_{CenterPeakSize} \\ \text{passed}, & \text{else} \end{cases} \tag{5.6}$$

The radius can be detected as a peak in the histogram R. R is indexed by i where i represents the distances of edges used to the estimated center C_c. Similar to the center test, T_r will be used to judge the shape measurement. Equation (5.7) shows when the T_r

[5]The draw function adds the line and will not delete existing lines.
[6]*blob.cc* is defined in Section 2.3.5.



a) Source

b) Edge image

c) No filter

d) Filter 1: Color-based

e) Filter 2: edge direction based

f) Filter 1 and 2

Figure 5.5: The graphs (c) - (f) show peaks formed by the lines L_i in the Hough space corresponding to the sphere of figure (a) and (b) with different filters applied.

fails.

$$T_r = \begin{cases} \text{failed}, & \text{if } \Sigma\ PeakSize < \epsilon_{MinPeak} \\ \text{failed}, & \text{if } \frac{PeakSize}{\Sigma\ Edges} < \epsilon_{RadiusPeakSize} \\ \text{passed}, & \text{else} \end{cases} \tag{5.7}$$

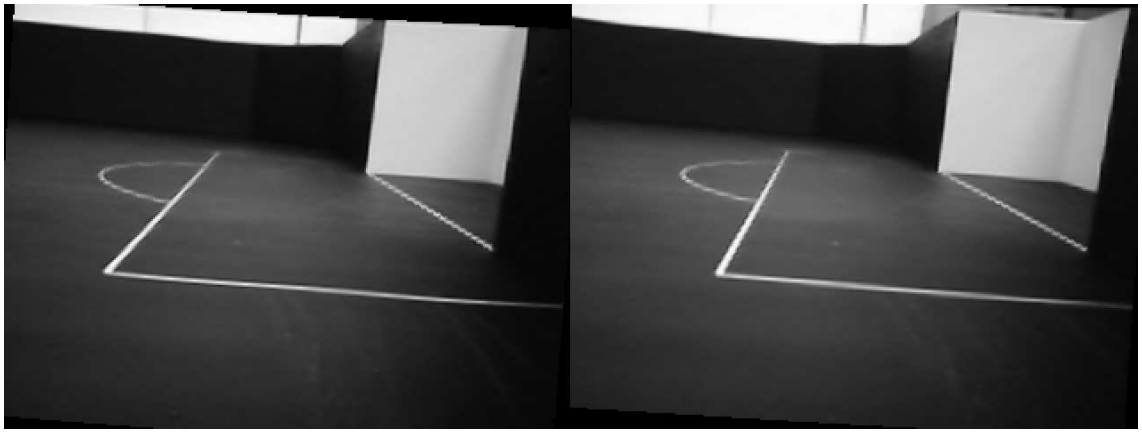
5.2.5 Rectification

Matlab[7] was used to calibrate the cameras and to create lookup tables for the rectification. Section 3.4 shows how images can be rectified by using a preprocessed lookup table. Matlab includes a vision library which generates eight lookup tables per image. The functions for generating those tables are `claib_gui` for finding the camera properties and `stereo_gui` for generating rectified images. Matlab names the tables `index_1_left - index_4_left`. Those tables contained the information to where a pixel should be moved from the source image to the calibrated target image and the tables `mulit_1_left - mulit_4_left` containing the a scale factor for every pixel moved. The tables are in the size of the image width $\times$ the image height. The result of four lookup tables is a smooth image. The implementation on the target system uses only one of the generated lookup tables to reduce the processing time caused by random SDRAM access. The result is a rectified image but less smoother than the Matlab-rectified ones. Figure 5.6 shows two rectified images where the top one was created by the Matlab rectification and the bottom one by the algorithm on the DSP with only one lookup table per image.

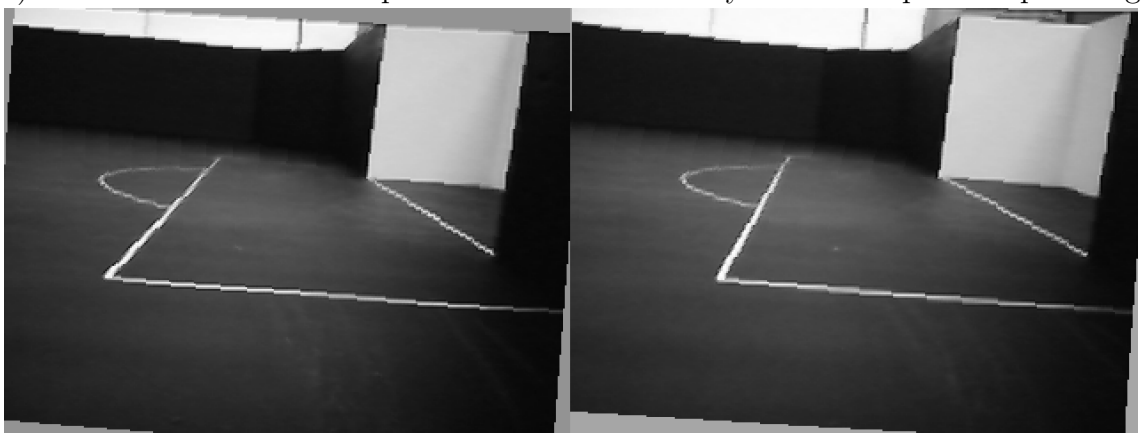
5.2.6 Object Detection

The implemented object detection can be used for different types of objects like rectangles, spheres, boxes and so on. But the aim in this thesis was to build a general detection concept and to implement detection for spherical objects. Other shaped objects are currently only detected by their color.
Figure 5.7 shows the implemented general interface for detecting objects. Objects are defined in an object model with their dimension and color. The detection concept is based on different tests where every test result affects for the accuracy of the detection. A positive test votes that the object belongs the the current model. In the first step, the object colors are used to segment the image and to find blobs. The dimension test is used to remove small or unformed blobs. (Example: A blob with a small height and long width will not be used even if the color indicates a relation to a model). The area test tries to match the blob area with the related model shape. Section 2.2.2 shows such a test for circles. The next two processing steps are then used to measure the shape of the object and its center. If a blob passes all tests, a related object will be stored with its position in the object history. If the detection fails on a test, the object will be stored with a lower accuracy.

[7] www.mathworks.com



(b) Rectified with four lookup tables and four intensity scale lookup tables per image



(b) Rectified with one lookup table per image

Figure 5.6: Two rectification techniques by using one or four lookup tables.

The accuracy indicates then how many properties of the detected objects can be used. An accuracy under a certain threshold indicates that only the estimated direction to the object can be used. The mode and settings define thresholds on the detection and the camera properties like resolution and the threshold for the dimension test.
We can see in Figure 5.8 that not all spheres are detected with their size but the direction to them was estimated. This gives the world model repository on the robot the possibility to complete or verify its current world model.
The distance to an object is estimated by its size and appearance in the image. Both techniques are discussed in Section 4.1.1 and Section 4.1.2. The distance estimation by the relative position to the floor is of high interest for rectangles because the shape in the image changes strongly with the point of view.
The stereo system is used to verify the detection and to raise the accuracy of a detection. If the object was seen in both images, the average between them is used to create the final object for the object history. The object history is transmitted via SPI after every cycle to the strategy unit.

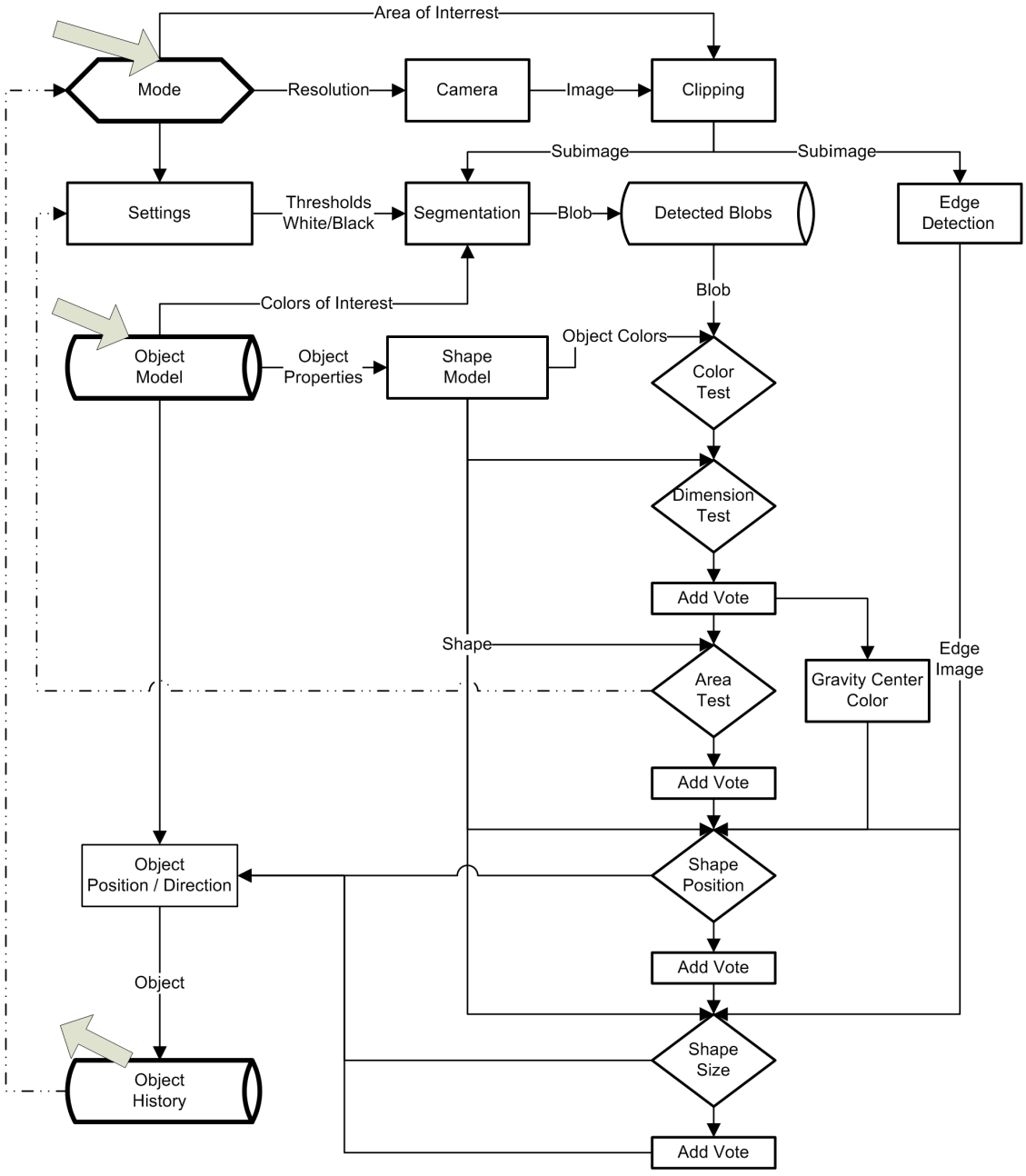


Figure 5.7: Data flow of the object detection. The large arrows mark the input and output.

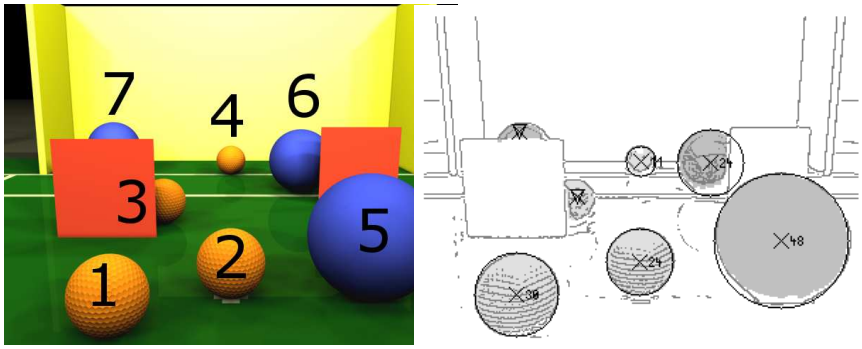


Figure 5.8: Left, computer generated test image to detect circles. Right, related edge image and the detected circles.

5.2.7 3D Lines

The distance to detected lines is estimated by searching the corresponding line in the other image. The nearest horizontal line with a similar angle is chosen to be the corresponding line. This was implemented by computing the x_m and y_m position of the line mi-point. A corresponding candidate in the other image is a line where the two end-points are above and below y_m. Horizontal and nearly horizontal lines are excluded from the 3D line detection. The orientation of the line in the three-dimensional space is computed by the disparity $d0$ and $d1$ of the line A for the start- and end-point which are therefore defined by (5.8) and (5.9).

$$d0 = x0_B - x0_A \tag{5.8}$$

$$d1 = x1_B - x1_A \tag{5.9}$$

The disparities are computed by using the slope s of the corresponding line described in Equation (5.10) - (5.12). This is done because occlusions and/or problems in the line detection can cut lines into more segments. Therefore, the detected line segments on the line have, in the two images of the stereo set, different start- and end-points but the line equation is still the same. Figure 5.9 shows a line in the three-dimensional space and its projection on the left and right image plane. The gray shadow in the left image represents a copy of the line in the right image.

$$s = \frac{x1_B - x0_B}{y1_B - y0_B} \tag{5.10}$$

$$d0 = (s * (y1_A - y0_B) + x0_B) - x0_A \tag{5.11}$$

$$d1 = (s * (y1_A - y1_B) + x0_B) - x1_A \tag{5.12}$$

The actual position in SI units is computed by using the equations (3.16) and (3.17) of Section 3.3.1.

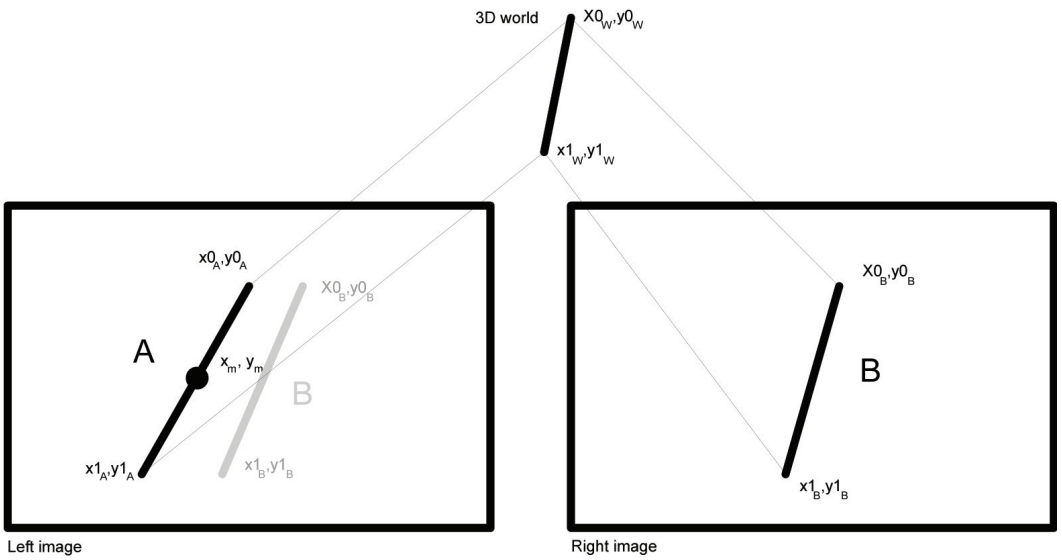


Figure 5.9: A 3D line and its corresponding projections in the left and right image.

5.2.8 3D Edges

Next to the 3D lines there are also 3D edges computed to detect undefined obstacles in the driving direction. We assumed that the images are rectified and searched only in the horizontal epipolar line for a corresponding edge with the edge gradient direction. The nearest matching edge has always been chosen as the corresponding edge. In addition thresholds were used for a minimum and maximum disparity.

5.2.9 Summary

We covered the implementation of the vision system. The result of the vision system can be used to find the goal, the robots and the game ball by using the result of the object detection. The 3D lines provide the basis for a self localization based on the lines on the playing field as well as for detecting rectangular structures.
Unexpected objects can be avoided by using the cloud of 3D edges. The stereo system on one hand is used to extract 3D lines and edges and on the other hand to verify the detected objects on their second appearances in the other image. The next chapter will talk about the results based on the implementation described in this chapter.

Chapter 6

Results

"If it doesn't fit, get a bigger hammer."

(Richard A. May)

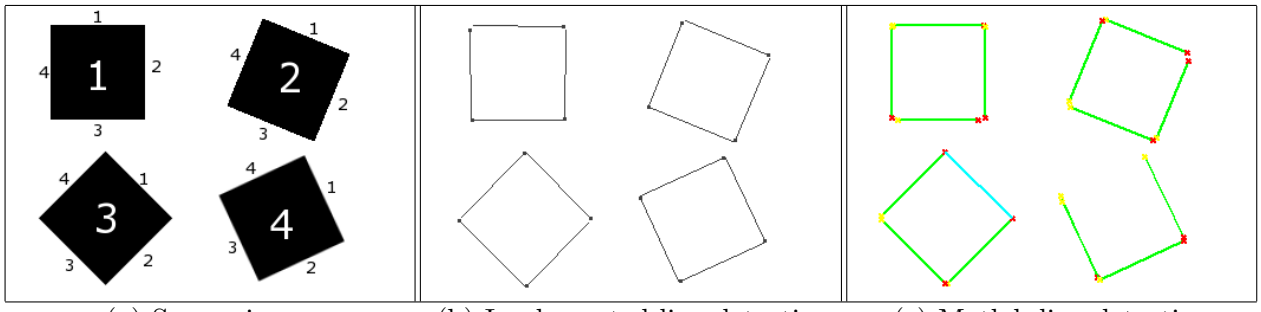
Results in this chapter present the capability of the implemented vision sensor. Section 6.1 to 6.3 discuss tests on images of synthetic and arranged real scenes. Those images are used to test the line detection, the feature-based stereo vision, the blob and the ball detection. Section 6.4 focus on the detection of objects on the playing field in the test environment. Therefore, images were taken in real situations in order to discuss different problems like lighting conditions and hidden objects. The usability of the vision data for a self localization is shown in Section 6.5. Timing results of the algorithm conclude this chapter followed by a summary.

6.1 Line Detection

Let us compare the line detection algorithm with the Hough transform line detection algorithm of Matlab 7.0.1[1]. We used a white image with black rectangles to find problems in the detection of lines in certain angles. Figure 6.1 shows the black rectangles rotated in $\frac{\pi}{8}$-angle steps and the detected lines of the implemented algorithm as well as lines detected by the Matlab algorithm. We can see in Figure 6.1.b that all lines were detected by the implemented line detection algorithm but not by Matlab algorithm in Figure 6.1.c. The reason why the line segment 4-4 in Figure 6.1.c has not been detected in this scenario is found in the Matlab function `houghpeaks`. `Houghpeaks` searches in the Hough space for a

[1]The code for the Matlab line detection used was provided by the Matlab help in the chapter *Image Processing Toolbox User's Guide to find lines in an image.*

given number of maximum peaks which are related to lines. In our case, the value was set to 16. Tests showed that 16 was the best value to detect lines in the test image without a double detection of a line.



(a) Source image (b) Implemented line detection (c) Matlab line detection

Figure 6.1: Line detection test on a 2D image with rectangles in different angles.

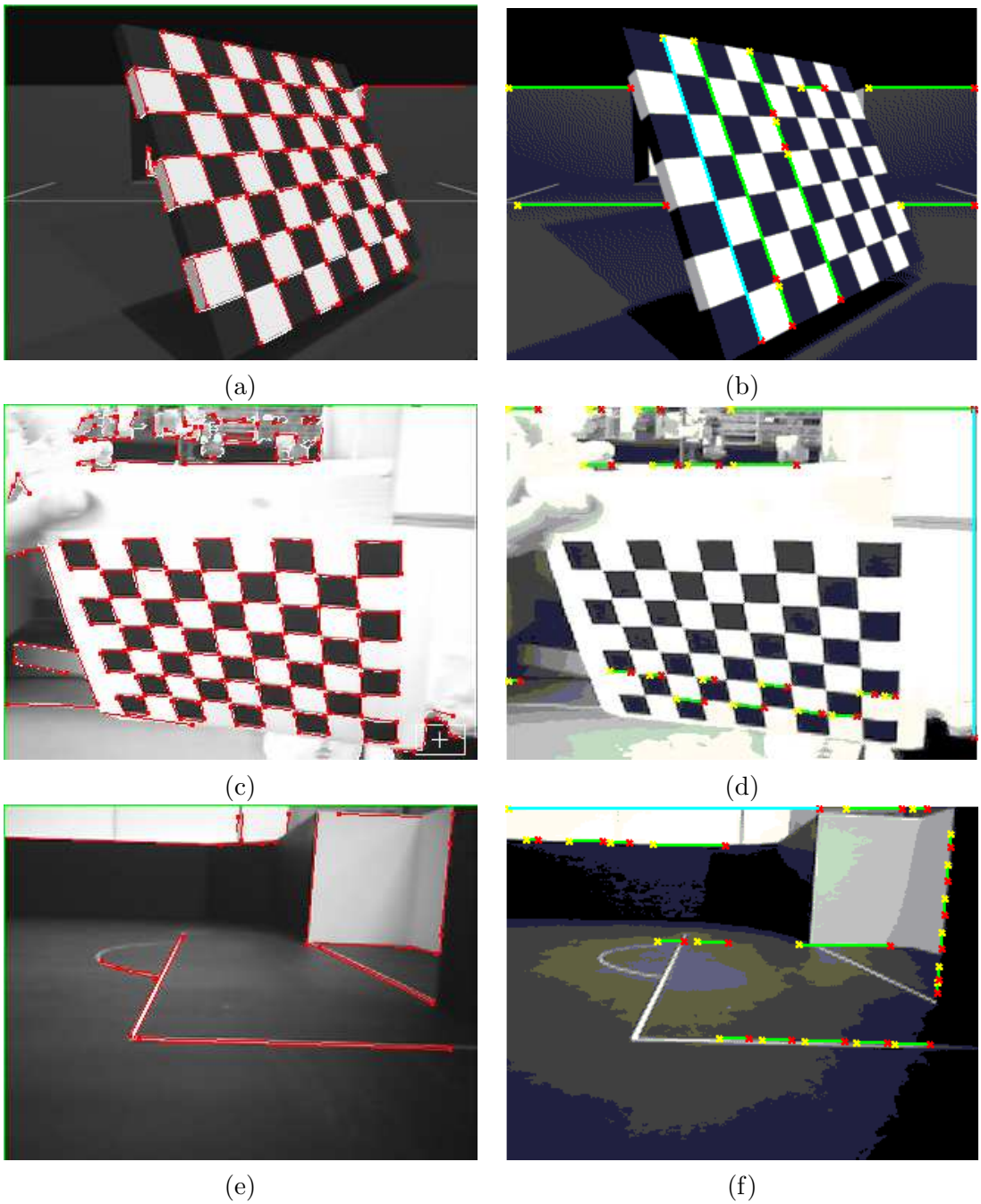
The Table 6.1 shows the start x_0, y_0 and end x_1, y_1 point of each line caused by the black rectangles in Figure 6.1.a. The "Difference" column represents the difference between the original line segment $x_0, y_0 - x_1, y_1$ and the detected line segment $\acute{x}_0, \acute{y}_0 - \acute{x}_1, \acute{y}_1$ by using the Equation (6.1).

$$\text{Difference } = \sqrt{(x_0 - \acute{x}_0)^2 + (y_0 - \acute{y}_0)^2}, \sqrt{(x_1 - \acute{x}_1)^2 + (y_1 - \acute{y}_1)^2} \tag{6.1}$$

The "Average" row in Table 6.1 indicates the average mismatch in pixels. We can see that the quality of implemented line detection is with a 2.11-pixel average difference minimally better than the average difference of the Matlab line detection with a value of 2.21 pixels. Another benefit in the implemented line detection algorithm is the correct detection of the line direction. The start point of the lines are marked with a dot in the implemented line detection algorithm in Figure 6.1.b and with a red cross in Matlab algorithm in Figure 6.1.c. The detected lines should be oriented in a loop around the rectangle. This is the case in the implemented line detector but not in the Matlab line detection.
Figure 6.2 shows real and synthetic images with the implemented algorithm on the left and the Matlab line detection on the right. We see in images in Figure 6.2 that the implemented algorithm performs better on short lines than the Hough transform algorithm of Matlab.

		Implemented line detection		Matlab line detection	
Line	Original line	Detected line	Difference	Detected line	Difference
1-3	25,16 - 104,16	26,17 - 103,17	$\sqrt{2}, \sqrt{2}$	27,17 - 104,17	$\sqrt{5}, 1$
1-2	104,16 - 104,95	105,20 - 105,95	$\sqrt{18}, 1$	105,18 - 105,95	$\sqrt{5}, 1$
1-3	104,95 - 25,95	102,97 - 26,96	$\sqrt{8}, \sqrt{2}$	99,97 - 31,67	$\sqrt{29}, \sqrt{40}$
1-4	25,95 - 25,16	25,95 - 24,19	$0, \sqrt{10}$	26,95 - 26,18	$1, \sqrt{5}$
Average			1.93 pixel	-	2.55 pixel
2-1	205,11 - 277,41	206,12 - 276,41	$\sqrt{2}, 1$	207,12 - 276,40	$\sqrt{5}, \sqrt{2}$
2-2	277,41 - 248,113	277,43 - 249,113	$\sqrt{4}, 1$	277,47 - 250,112	$\sqrt{4}, 1$
2-3	248,113 - 175,84	246,114 - 178,87	$\sqrt{5}, \sqrt{18}$	247,114 - 177,86	$\sqrt{2}, \sqrt{8}$
2-4	175,84 - 205,11	175,83 - 203,13	$1, \sqrt{8}$	176,81 - 204,13	$\sqrt{10}, \sqrt{5}$
Average			1.96 pixel	-	2.04 pixel
3-1	71,123 - 127,179	73,126 - 127,179	$\sqrt{13}, 0$	71,124 - 128,180	$1, \sqrt{2}$
3-2	127,179 - 71,235	123,185 - 72,236	$\sqrt{52}, 1$	128,180 - 73,235	$\sqrt{2}, \sqrt{4}$
3-3	71,235 - 15,179	70,236 - 15,181	$\sqrt{2}, \sqrt{4}$	71,235 - 17,181	$0, \sqrt{8}$
3-4	15,179 - 71,123	15,179 - 67,125	$0, \sqrt{20}$	17,178 - 71,124	$\sqrt{4}, \sqrt{1}$
Average			2.46 pixel	-	1.48 pixel
4-1	239,126 - 273,198	240,128 - 273,198	$\sqrt{5}, 0$	240,128 - 273,190	$\sqrt{5}, -\sqrt{64}$
4-2	273,198 - 201,232	272,200 - 202,232	$\sqrt{5}, 1$	273,199 - 202,232	$1 - 1$
4-3	201,232 - 167,160	200,232 - 168,162	$1, \sqrt{5}$	200,230 - 169,161	$\sqrt{5} - \sqrt{5}$
4-4	167,160 - 239,126	168,160 - 238,128	$1, \sqrt{5}$	-	-
Average			2.12 pixel	-	2.78 pixel
Average			2.11 pixel	-	2.21 pixel

Table 6.1: Errors of the implemented line detection algorithm.



(a) (b) (c) (d) (e) (f)

Figure 6.2: Results of the implemented line detection algorithm and the Matlab Hough transform line detection algorithm. The left column presents the results of the implemented line detection and the right column the results of the Matlab line detection.

6.2 Line Based Stereo Vision Algorithm

The performance of the line-based stereo vision system on vertical lines is shown in this section. Synthetic and non-synthetic images have been used to test the algorithm. The synthetic images were created with *3ds max*[2] with boxes in different distances to the camera. The left and right border of these boxes were used as vertical lines for the test. The setup for the synthetic images used two virtual parallel cameras with a focal length of 311mm and a base line length of 30mm. The real images were taken by the robot in the RoboSot league soccer environment. Baseline length and focal length were measured by the calibration algorithm of Matlab[3]. The images showed after the rectification a focal length of 386mm and a base line length of 30.47mm. For the synthetic images, no calibration was necessary because the cameras were already placed coplanar in the virtual environment.
The test images are shown in Figure 6.3. We see in the images a box placed between the optical center of the two cameras of the stereo camera system. The distance of the box to the camera setup varies between 250mm on the top left image to 1000mm at the bottom right image. The distances are listed in column d_r in the Table 6.2. The Table 6.2 includes three columns for the real and three columns for the synthetic images. The column d_{left} represents the measured distance to the left border detected as a line on the box. d_{avr} stands for the average of distance between the left and right lines and the column $|\frac{100\cdot(d_r-d_{avr})}{d_r}|$ represents the absolute error of the real distance in percentages.

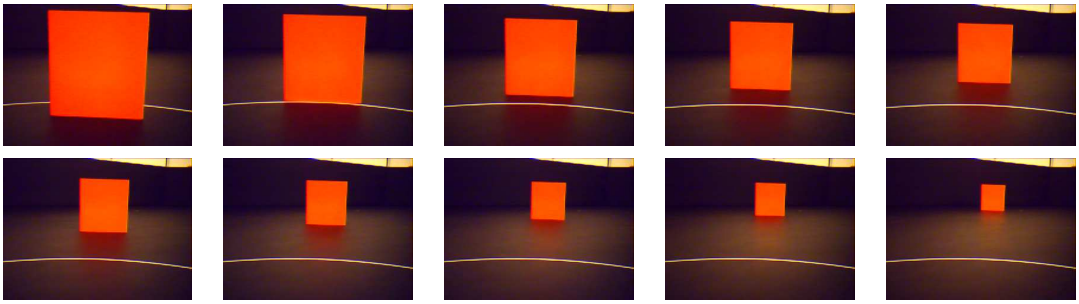
	Real Data			Synthetic Data		
d_r [mm]	d_{left}[mm]	d_{avr}[mm]	$\lvert\frac{100\cdot(d_r-d_{avr})}{d_r}\rvert$[%]	d_{left}[mm]	d_{avr}[mm]	$\lvert\frac{100\cdot(d_r-d_{avr})}{d_r}\rvert$[%]
250	220	228	8.8	252	253	1.2
300	280	282	6	301	307	2.33
350	342	333	4.86	352	347	0.86
400	428	407	1.75	407	404	1.25
450	519	494	9.78	569	496	10.22
500	582	567	13.6	586	594	18.80
600	805	801	33.34	691	704	17.33
700	-	-	-	812	794	13.43
800	-	-	-	-	-	-
1000	-	-	-	-	-	-

Table 6.2: 3D line detection error related to Figure 6.3.

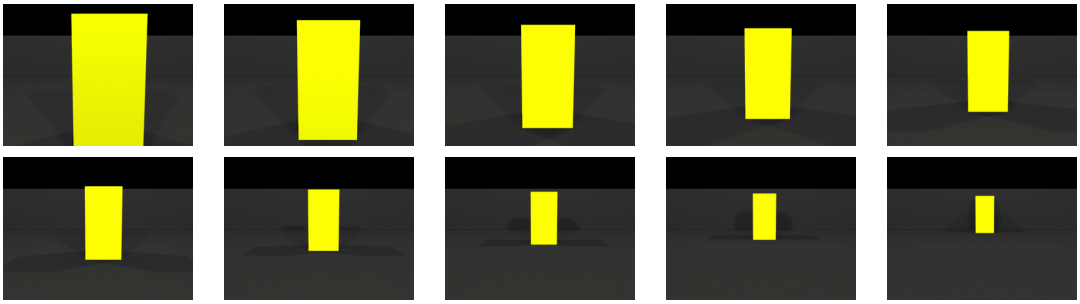
We see in Table 6.2 that lines beyond a distance of 500mm cause errors higher than 10% of the real distance to the line. Lines beyond 700mm are not always detected with depth information.
Figure 6.4 shows the distance to a line in relation to the detected disparity and the

[2]3ds max is a software product to create photo-realistically synthetic scene. Details can be found at www.discreet.com and www.autodesk.com

[3]Matlab 7.0.1 provides a function called `stereo_gui` to rectify and calibrate stereo image pairs.



(a) Real images



(b) Synthetic images

Figure 6.3: Test images for evaluating the 3D line accuracy.

distance aberrance if the disparity is detected incorrectly. We also see in Figure 6.4 that an incorrect detection in the line's disparity causes a non-linear distance error. This error increases indirectly proportional to the disparity. The relative error on an incorrect disparity detection is printed in the two discontinuous lines in Figure 6.4.

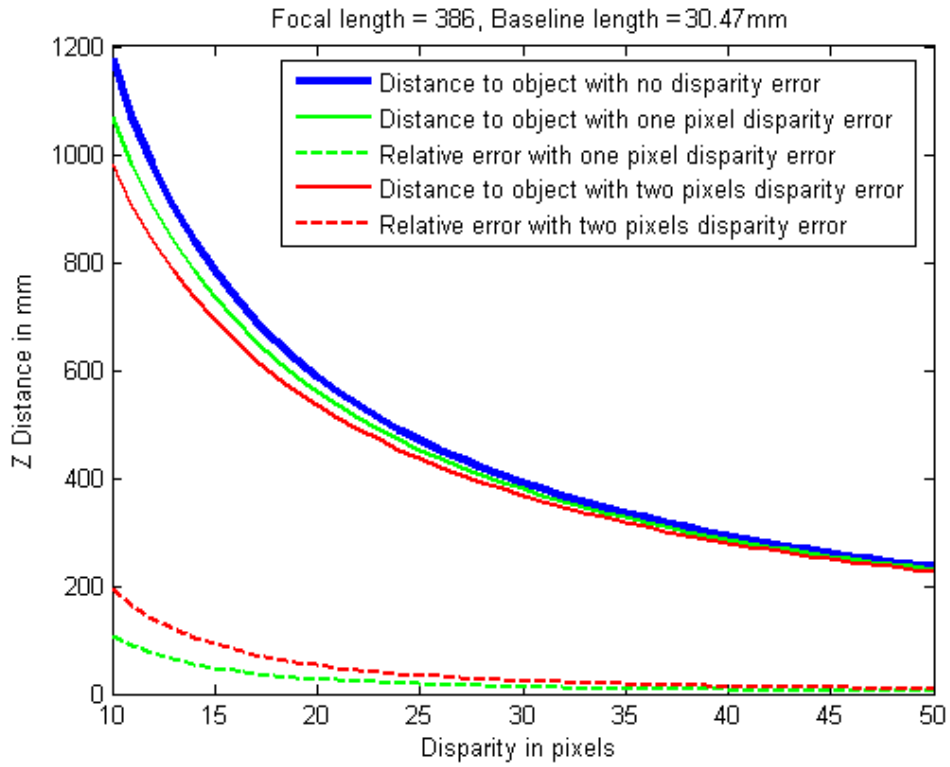


Figure 6.4: Distance in relation to a given disparity value

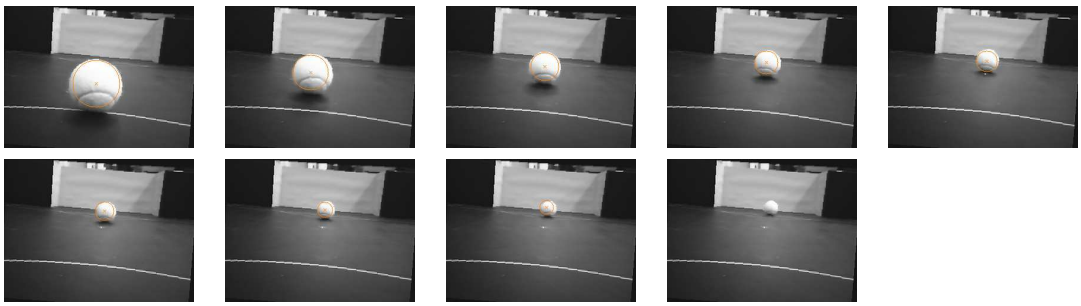
6.3 Object Detection

The detection of objects in the surrounding area of the robot is one of the essential functional requirements. The goal and the game ball must be detected if they are in the field of view of the vision system. Recognizing the game ball and estimating its position is essential to play robot soccer. The detection of the directions to the goals allows a rudimentary self-localization to find the playing direction of the robot soccer game. The following section presents results of the position estimation of the game ball and of the direction estimation toward the goals.

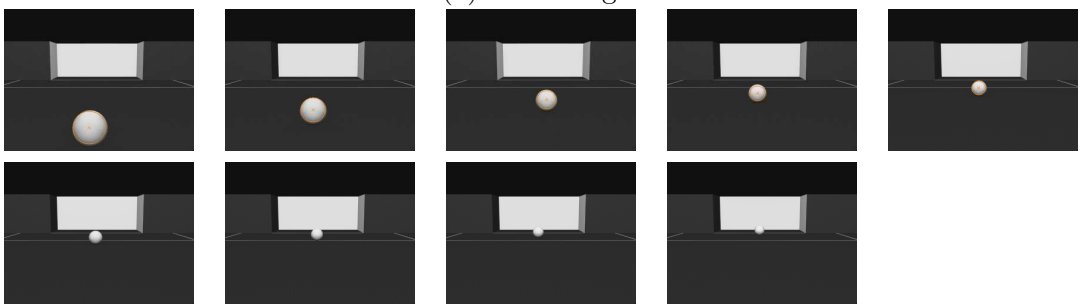
6.3.1 Ball Detection

Tests have been performed to find the accuracy of the game ball detection on synthetic and real images. A game ball, in this case a yellow tennis ball, is placed in front of the robot in different distances. Figure 6.5.a shows real images taken by the robot on the playing field. Focal length and baseline distance are the same as in Figures 6.3 of Section 6.2. The nine images of synthetic data in Figure 6.5.b represent the same scene as in the real images,

but they are generated using a different focal length in 3ds max. The exact distances to the ball are printed in Table 6.3, where the rows in Table 6.3 correlate in order with the images in Figure 6.5.



(a) Real images



(b) Synthetic images

Figure 6.5: Test images for evaluating the accuracy of the object detection algorithm.

In Figure 6.5 we see that the ball appears smaller in the synthetic images (b) than in the real images (a). The shorter focal length in the synthetic camera causes this effect. This is the reason why the ball has not been detected beyond 800mm in the synthetic images but has been recognized in the real images. Two images from the left and right camera are used to estimate the distance to the ball.
d_{right} and d_{left} represents the distance estimated by the right and left camera. d_r in Table 6.3 represents the real distance of the object. The detected distance of the ball to the camera can be found in the columns d_{left} and d_{right}. The fourth and seventh column represent the difference to the real ball in percent by using the average distance $d_{avr} = \frac{d_{left}+d_{right}}{2}$ between the left and the right detection in the images of the stereo vision system. The distance estimation is sufficient enough to measure the distance between ball (6cm in diameter) and camera up to 800mm with an error less than 10% of the real distance.
A distance of up to 800mm is enough for the robot to plan its next task as long as the direction to the ball can be estimated.

	Real Data			Synthetic Data		
d_r [mm]	d_{left} [mm]	d_{right} [mm]	$\lvert\frac{100\cdot(d_r-d_{avr})}{d_r}\rvert$[%]	d_{left}[mm]	d_{right}[mm]	$\lvert\frac{100\cdot(d_r-d_{avr})}{d_r}\rvert$[%]
300	295	282	3.83	290	287	3.83
400	399	374	3.38	407	398	0.62
500	515	483	0.2	502	480	1.8
600	589	579	2.67	586	592	1.83
700	727	610	4.5	711	667	1.57
800	824	724	3.25	829	778	0.44
900	-	827	8.11	-	-	-
1000	-	965	3.5	-	-	-
1100	-	-	-	-	-	-

Table 6.3: Object detection error related to Figure 6.5.

6.3.2 Blob Detection

This section discusses the results on the direction estimation to colored objects. The implemented shape detection algorithm is limited to circular shapes, but the vision system is able to recognize objects also by their color. The implementation includes a general interface for detecting objects even if shape detection for this object is not implemented. The detection is then only based on the object color. Figure 6.6 shows different possible objects in the robot soccer environment which are detected by their color. All robots have to wear colored marks on their body. This colored mark is detected also by the system as well as the soccer goals. We see on the images in Figure 6.6 that the shape has not

(a) (b)

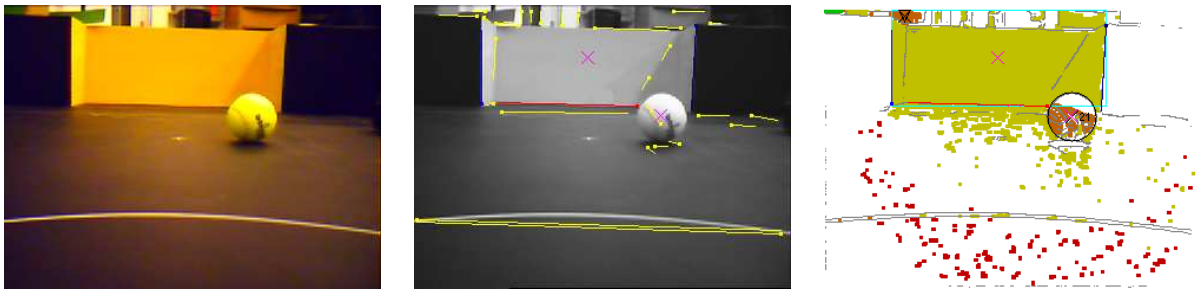
Figure 6.6: Test images for evaluating colored object detection

been detected but the direction to the object was recognized and drawn as a box. The boxes in the images are related to the detected blob with its center marked with a "+" and the related direction to the object marked with a "×" which represents the color center

of mass[4]. Figure 6.6.a also shows a common problem for the detection of blobs. The reflections of the robot's color mark on the playing field has caused an incorrect detection of a blob with the same color as the mark on the robot. But the world model repository[5] filters objects which appear on physically impossible places. A reflection of a ball or the refection of the robots color mark indicates objects with an estimated position under the playing field. Such reflected objects are filtered by the world model repository.

6.4 Objects on the Playing Field

Presented in this section are results of real images taken by the robot during a game. For this reason, a rectangle shape detection algorithm which is not discussed but implemented is used to find and measure rectangles in the image. This algorithm finds the corresponding border lines of a detected rectangle. The related lines of a rectangle are than delivered to the world model repository. Figure 6.7 shows a scene where the game ball lies next to the goal. The ball was detected and measured as well as the goal. The half viewed circle landmark on the floor in Figure 6.7.b causes a problem. The circle appears as a line in the detection. Circular landmarks exhibit general problems in the detection because of the camera position on the robot which is only 12cm above the ground and views at 10° downwards. A circular landmark appears generally as flat ellipse segments because of the camera position on the robot.
The goal line has been detected correctly. Lines drawn in light and dark blue mark the detected goals on the left and the right side. The detected bottom line is colored in red. The shown cyan box represents the finally goal object recognized in Figure 6.7.c.
On the left side above the goal is a blob which does not belong to the playing field but we see on the "▽" that its assignment to a model failed because of the blob's deformed shape or blob dimension[6].

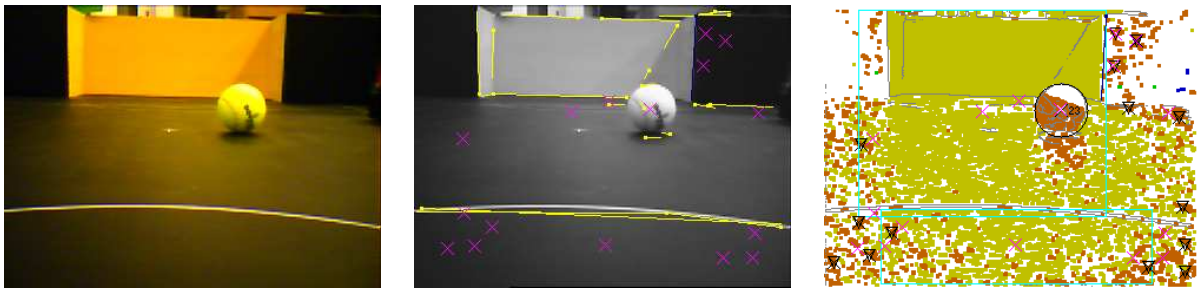


(a) (b) (c)

Figure 6.7: Detected blob on the background and goal borders.

[4]Color center of mass is described in Section 2.3.5.
[5]The world model repository is described in Section 5.1.1.
[6]Blob dimension and shape test is discussed in Section 5.2.4.

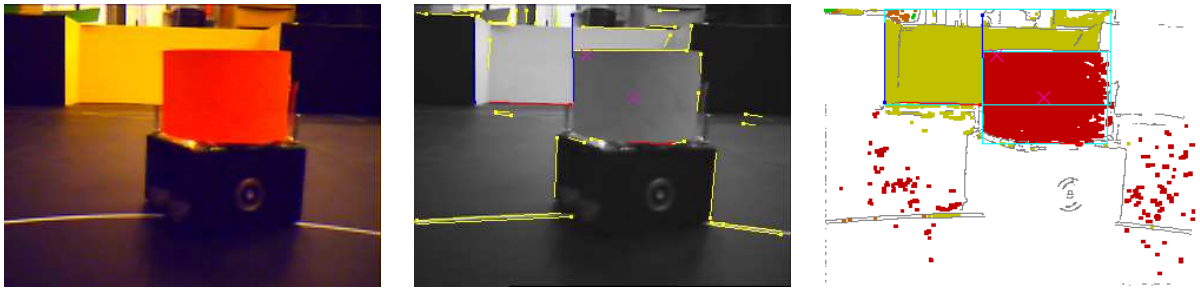
Wrong camera settings like shutter speed, gain and black/white balance produce a faulty color blob detection. Figure 6.8 shows the first image taken after the camera initialized with correct camera settings. The camera needs five to ten images to respond to the initialized setting. Therefore, the first images of the camera after a camera initialization are, because of this effect, not usable for the object detection algorithm presented. We see in the source image (a) in Figure 6.8 the drift toward yellow in the color spectrum. This effect becomes clearer in the blob detection in Figure 6.8.c. Nearly half of the playing field was detected there as a blob referring to the yellow goal.



(a) (b) (c)

Figure 6.8: Camera initialization problem

A detected robot is shown in Figure 6.9. A distance estimation to the robot at the detected width of the colored mark is not possible because the robot diameters are only defined with a maximum value. The game rules determine that the colored mark on a robot must be visible from all sites starting from 7cm to 12cm above the ground. Thus, the detected bottom and/or top line of the rectangle are useable features. The detected height of the mark on the robot and/or the position of the top or bottom line of the mark in the image are used for an estimation of the robot's location.
Another problem is seen in Figure 6.9. The detection of the left border line of the color mark of the robot is longer than the actual mark. The structure of the background[7] interferes with the detection of the border line.

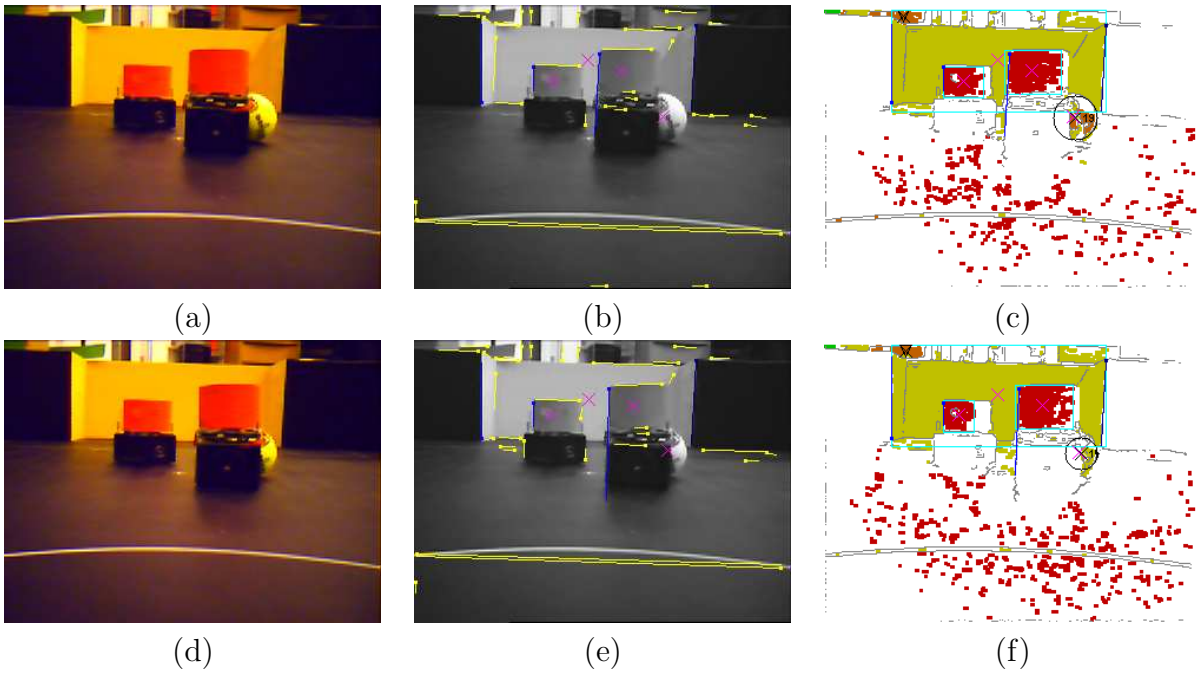


(a) (b) (c)

Figure 6.9: Player detection by its colored mark

[7]Background means in this case the viewed structure which does not belong to the playing field.

Figure 6.10 shows two problems: one with the ball detection and another one with the edge detection. First is the ball detection problem. We see in the bottom row of the images that the detection of the circular object (game ball) was not correct. One robot hides more than half of the game ball. This causes problems in the detection. The radius is wrong because edges in the surrounding area are detected as part of the circular shape. Normally, if the surrounding edges are not detected as part of a circle, the algorithm will fail at the shape/circle center test[8] and the algorithm will mark the object with low detection accuracy. A low accuracy indicates that only the estimated direction to the object is usable. This information is essential for the world model repository. Figure 6.10.c shows a correct detection of the game ball.



(a) (b) (c)

(d) (e) (f)

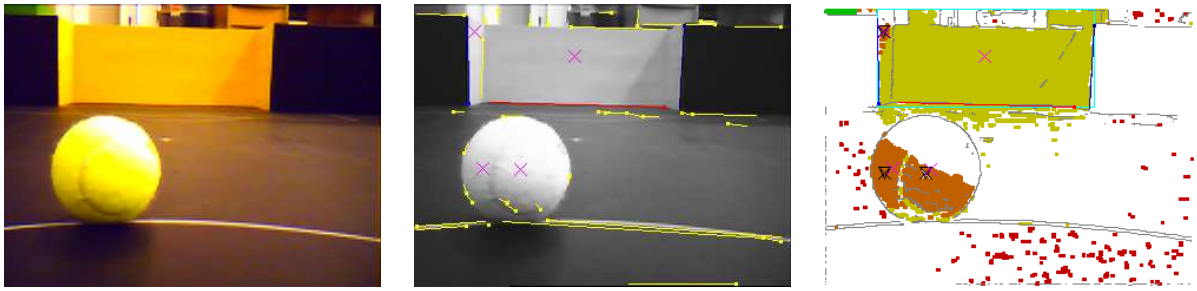
Figure 6.10: Hidden ball.

The second problem occurs on the edge detection in images (c) and (f) in Figure 6.10. The edge detection failed with the right robot on the right border of the color mark. Figure 6.11 shows the problem in detail. The reason lies in the luma information[9] in this area which, in this case, does not change between the red and the yellow color. A visible change for the human eye in this area is only based on the chroma information. The edge detection, which is based on the luma information, does not find the obvious edge for us humans. This effect has been observed in many cases on red and yellow objects on the side facing the window side of the laboratory which could not be darkened.
Stuctures on the game ball are the reasons why the detection of the sphere fails in Figure 6.12. One league of the FIRA robot soccer community uses a yellow tennis ball as the

[8]The shape center test was discussed in Section 5.2.4.
[9]The luma and chroma information of a YUV image is described in Section 2.3.4.

Figure 6.11: Edge detection problem.

game ball. The typical structure or fabrication line on the tennis ball separates the blob, which belongs to the ball, into two parts. The circle detection searches only in a limited area around the blob for a circular arc and fails. The results shown are two blobs related to the ball model with low detection accuracy.
The Figure 6.12 shows a welcome feature for a non jet-implemented landmark recognition. White lines on the playing field are detected with two lines going in opposite directions. This attribute is usable in the next version of the implementation to associate lines to landmarks. We have to be careful with this feature because as we see in Figure 6.7, the line detection fails on flat circular arcs. A second line in the opposite direction is missing if the landmarks are far away like in Figure 6.14. An object which is too close to the

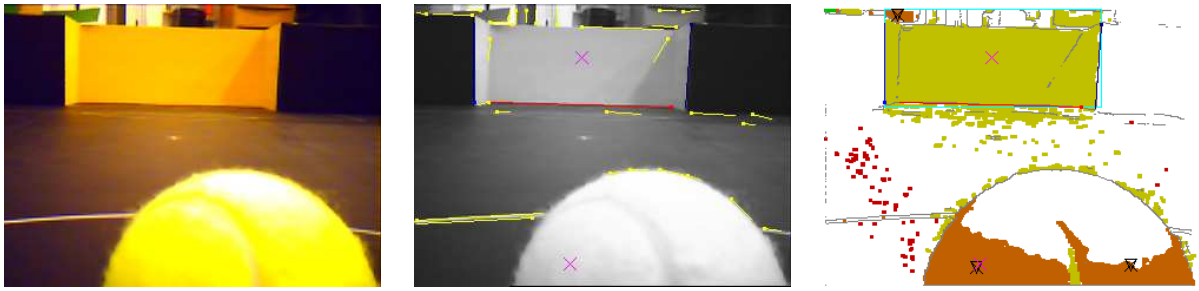


(a) (b) (c)

Figure 6.12: Goal seen from a sharp angle.

camera appears in another color or is too brightly. This effect is seen in Figure 6.13. The color on the top of the spheres appears too brightly. Therefore, the top of the ball is not detected as a known area. Even if the blob is detected correctly we face another problem. The circle shape detection will fail because only a part of the game ball is visible in the image. A solution to this problem has not yet implemented. Figure 6.14 shows a problem if the goal is seen from a sharp angle. The distance estimation to the goal by its size[10] fails. The projection of the goal in the image is not rectangular. But the detected bottom line allows us to estimate the position because we know the camera position relative to the

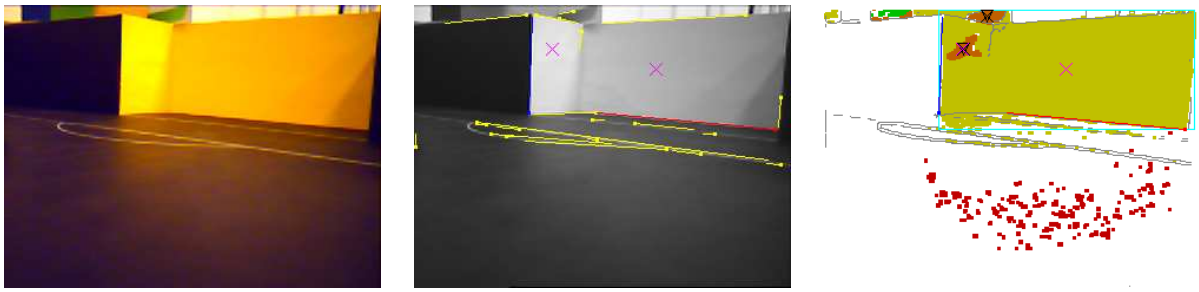
[10]Distance estimation by known object size is discussed in Section 4.1.1



(a) (b) (c)

Figure 6.13: Goal seen from a sharp angle.

playing field. We can use the technique discussed in Section 4.1.2 to estimate the goal's position. An additional error causes the assumption in the implementation that the goal is a rectangle. This is not true, because the goal is a box which is open on two sides. For this reason, the vision system has been designed to deliver, on a rectangle detection, not only the estimated position of the rectangle to the world model repository, it also submits the direction to the detected border lines. These inputs are then used in the case of a detected goal to estimate the robot's location by using a particle filter. Further discussions on the particle filter and other localization methods can be found in [FT06].



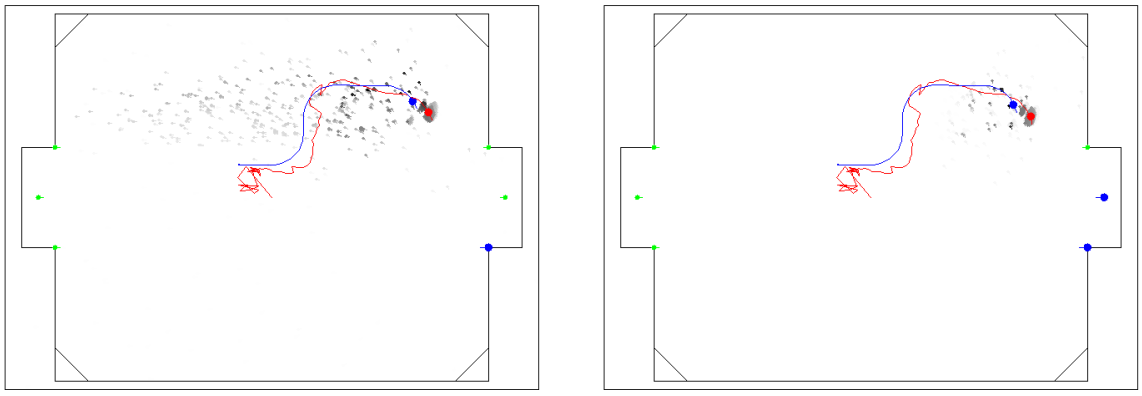
(a) (b) (c)

Figure 6.14: Goal seen from a sharp angle.

6.5 Self-Localization

The simulation done in [Deu07] uses a particle filter where the split information of an object in object features (goal post and goal bottom line) is used for self localization.
Figure 6.15 shows the localization process in time[Deu07]. Data from odometry and vision are used for the localization. Gray dots in Figure 6.15 are particles where each one represents the probability that the robots reside on this location on the playing field. The final location (red dot on the end of the red line) is estimated by a weighted average of all

particles where the particle probability is the weight.
Figure 6.15.a shows a swarm of particles denoting possible locations of the robot, according to previously done estimation in combination with the current vision sensor data (right goal post). The swarm is shaped like a comet's tail because the goal post feature is only detected with its direction. In the next step, the bottom line of the goal is detected additionally to the goal post which is seen in Figure 6.15.b. The swarm concentrates on a small area close to the real position. This increase of quality is not only because two features are visible but also because the distance to the bottom line of the goal is known.



(a) (b)

Figure 6.15: Particle filter of the self localization.
Images are taken from [Deu07]

6.6 Timing

Tests have been done on images taken by the robot in the playing field. The processing time for an image depended on the chosen area of interest[11]. The average processing time of images shown in the last sections was at 220ms (4.5Hz) on a resolution and an area of interest of 320x240pixels. The Table 6.4 shows the runtime of the algorithm on the robot with the processing time in every processing step for real images. The following list describes the functions printed in the first column of Table 6.4:

- **Figure:** Describes to which image the timings belong in this thesis.
- **Edge detection:** Includes the process to find edges with the canny edge detector and the process to reduce the image to 32 colors.
- **Open & Closing:** Process to reduce noise (see Section 2.3.1).
- **Line detection:** Process to find lines and depth information based on lines.

[11] Area of interest: This area represents an area in the image where the detection will be applied.

- **Shape detection:** Process to detect rectangles and circles near blobs.
- **Object detection:** Process to verify the detected objects with the information from the second camera.

Figure	6.7	6.10 top row	6.13	Average
Edge detection	89 ms	89 ms	85 ms	≈ 87 ms (40 %)
Open & Closing	26 ms	26 ms	27 ms	≈ 26 ms (12 %)
Blobs detection	70 ms	75 ms	60 ms	≈ 68 ms (30 %)
Line detection	20 ms	22 ms	15 ms	≈ 19 ms (9 %)
Shape detection	19 ms	18 ms	18 ms	≈ 18 ms (8 %)
Object detection	1 ms	1 ms	1 ms	≈ 1 ms (>1 %)
Processing time:	225 ms (4.3Hz)	231 ms (4.3Hz)	206 ms (4.9Hz)	≈ 220 ms (4.5Hz)

Table 6.4: Timing of the algorithm.

We see that 40% of the processing has been used by the edge detection. The edge detection is the process that needs most of the processing time because of the read operations for the image on the SDRAM[12]. The SDRAM accesses are the reason for the processing time of the open & closing operation (12%) and the blob detection (30%). The detection results of the edge and blob detection are stored in the L1 memory which is more quickly accessible than the SDRAM [PH98]. The line and shape detection benefits from the access speed of the L1 memory which is shown in the average processing time of 19ms and 18ms (9% and 8% of the total processing time).
The edge, open & closing and blob detection always process the whole area of interest. Therefore, the processing time of these functions varies slightly with the context of the image. Line and shape detection are related to the viewed scene especially the shape detection. Their processing time varies with the context of the image as we see in the row "line detection" in Table 6.4. The circle detection showed, on test images with more than one game ball, a significantly longer processing time. The change in the processing time changed linearly for every ball on the playing field. The rectangle detection showed only slight changes on the processing time if more rectangles were present in the image.
The frame rate could nearly be tripled in tests where the resolution has been reduced to 160x120 pixels. It was approximated around 12 Hz but the quality of the object detection decreased. Tests have also been performed with a resolution of 360x240 pixel and a limited area of interest to 300x200 pixels around the center. In these particular case the frame rate increased into a level above 5 Hz.

[12]A SDRAM read operation is around ten times slower than a read operation on the L1 memory. Details can be found in Section 5.1.3.

6.7 Summary

This chapter points out that requirements of the system, as specified in Section 1.3.1, have been fulfilled. Colored spherical objects like the game ball and colored blobs related to the game goals are detected by the vision systems. This was shown in the results in Section 6.3. It was also shown in Section 6.2 that the feature-based stereo vision system is able to provide 3D lines for a not yet implemented landmark detection within 500mm in the viewing direction. Experiments in Section 6.4 on real images from robots on the playing field demonstrate problems in real soccer game situations.
An example of a self-localization simulation presented the usability of the vision data together with the odometry in order to estimate the robot's position on the playing field. The average frame rate of 4.5Hz lies under the desired frame rate of 5Hz, but tests in Section 6.6 illustrated that a frame rate up to 12Hz is reachable with changes in the area of interest.

Chapter 7

Conclusion

"The significant problems we face cannot be solved at the same level of thinking we were at when we created them."

(Albert Einstein)

This Chapter presents a discussion to show relationships among observed facts in the context of this thesis. Additionally, a final section poses ideas for future work in computer vision on embedded systems.

7.1 Discussion

Is the Tinyphoon robot able to play robot soccer with the presented vision sensor? That is the essential question we would like to discuss in this section. The presented frame rate of 5Hz is at the limit of the proposed requirements. A realistic game where the robot should fight state-of-the-art robots with vision systems above the playing field (MiroSot League and Middle-Size League) will end with a fiasco for the Tinyphoon robot. The game ball speed (in MiroSot up to 60km/h) makes the vision sensor unusable. The shutter speed of the camera and the processing time is too slow. But the new proposed league, AMiroSot, is an environment where the robot can face enemies with similar problems. The result will be a game with a slow average traveling speed ($\approx$0.15m/s) of the robots in the AMiroSot league.
The Tinyphoon robot already showed, at the FIRA World championship 2006 at Dortmund in the RoboSot league, that it is capable of matching other robots of his kind.
The results presented of the object detection are good enough to allow the robot to localize itself and to find the game ball as well as to detect other robots on the playing field.

The results of the object detection showed us that the usage of a stereo vision system increased the detection accuracy of objects. A game ball was detected within 800mm with an error less than 10% to the real distance.
A feature-based stereo vision system detects lines in three-dimensional space. Those 3D lines can be used in a next version of the implementation to detect landmarks on the playing field.
Problems are still present and reported in this thesis. Lighting conditions and reflections are only some of the reported problems. But none of them conclusively prohibited the robot from playing robot soccer.

7.2 Future Work

Additional shape detection algorithms are major issues for future implementations. Currently, goals and marks on robots are only detected as colored objects. An implementation of a box detection algorithm would raise the field of application.
The rectification lookup table in this thesis is used to rectify the two images of a stereo camera set. But as we saw in Section 5.2.5, the image quality suffers from the rectification. A suggestion is to apply the object and line detection algorithm on the unrectified images and then to rectify only the extracted features for the 3D line extraction and object verification.

Bibliography

[Ana05] Analog Devices, Inc. *ADSP-BF561 Blackfin Processor Hardware Reference*, revision 1.0 edition, July 2005.

[BAS+06] M. Bader, M. Albero, R. Sablatnig, J. E. Simó, G. Benet, G. Novak, and F. Blanes. Embedded real-time ball detection unit for the YABIRO biped robot. In *WISES'06, Fourth Workshop on Intelligent Solutions in Embedded Systems*, volume 4, pages 1–14, 2006.

[Bax94] G. A. Baxes. *Digital image processing: principles and applications*. John Wiley & Sons, Inc., New York, NY, USA, 1994.

[BBH03] M.Z. Brown, D. Burschka, and G.D. Hager. Advances in computational stereo. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 25(8):993–1008, August 2003.

[BFD05] P. Biber, S. Fleck, and T. Duckett. 3d modeling of indoor environments for a robotic security guard. In *Computer Vision and Pattern Recognition, 2005 IEEE Computer Society Conference on*, volume 3, pages 124–124, 20-26 June 2005.

[BH99] Alireza Bab-Hadiashar. *Data Segmentation and Model Selection for Computer Vision*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1999.

[BHR86] J. Brain Burns, Allen R. Hanson, and Edward M. Reisman. Extracting straight lines. *Transactions on Pattern Analysis and Machine Intelligence*, 4:425–455, 1986.

[Blu06] Bluetechnix Mechatronische Systeme GmbH, Waidhausenstr. 3/19, A-1140 Vienna. *Hardware User Manual CM-BF561 V1.2, V1.3*, 100-1211-561-01.2.2 edition, April 2006.

[BN98] D.N. Bhat and S.K. Nayar. Ordinal measures for image correspondence. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(4):415–423, April 1998.

[BT99] S. Birchfield and C. Tomasi. Multiway cut for stereo and motion with slanted surfaces. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE*

International Conference on, volume 1, pages 489–495vol.1, 20-27 September 1999.

[Can86] J Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6):679–698, 1986.

[CB03] Sharlee Climer and Sanjiv K. Bhatia. Local lines: a linear time line detector. *Pattern Recogn. Lett.*, 24(14):2291–2300, 2003.

[CD97] P. Corke and P. Dunn. Real-time stereopsis using FPGAs. In *TENCON '97. IEEE Region 10 Annual Conference. Speech and Image Technologies for Computing and Telecommunications, Proceedings of IEEE*, volume 1, pages 235–238, 2-4 December 1997.

[DA89] U.R. Dhond and J.K. Aggarwal. Structure from stereo - A review. *Systems, Man and Cybernetics, IEEE Transactions on*, 19(6):1489–1510, November-December 1989.

[DA92] U.R. Dhond and J.K. Aggarwal. Analysis of the stereo correspondence process in scenes with narrow occluding objects. In *Pattern Recognition, 1992 . Vol.1. Conference A: Computer Vision and Applications, Proceedings., 11th IAPR International Conference on*, pages 470–473, 30. August-3. September 1992.

[Dau03] Kerstin Dautenhahn. Roles and functions of robots in human society: implications from research in autism therapy. *Robotica*, 21(4):443–452, 2003.

[Dav92] E.R. Davies. Simple two-stage method for the accurate location of hough transform peaks. In *Computers and Digital Techniques, IEE Proceedings-*, volume 139, pages 242–248, May 1992.

[Deu07] Tobias Deutsch. *Self-localization*. Master's thesis, Vienna University of Technology, 2007.

[DH72] R. O. Duda and P. E. Hart. Use of the hough transformation to detect lines and curves in pictures. *ACM*, 15(1):11–15, January 1972.

[DJ00] Gregory Dudek and Michael Jenkin. *Computational principles of mobile robotics*. Cambridge University Press, New York, NY, USA, 2000.

[DRM03] A. Darabiha, J. Rose, and J.W. Maclean. Video-rate stereo depth measurement on programmable hardware. In *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, volume 1, pages 203–210, 18-20 June 2003.

[Ent05] H. Entner. *Real-time 3D Reconstruction for Autonomous Football Playing Robots Using a Feature Based Stereo Approach*. Master's thesis, Vienna University of Technology, 2005.

[FT06] Dieter Fox and Sebastian Thrun. *Probabilistic Robotics*. MIT Press, 2006.

[Gan84] Sudaram Ganapathy. Decomposition of transformation matrices for robot vision. In *Proc. Int. Conf. Robotics and Automation*, pages 130–139, 1984.

[Har94] R.I. Hartley. An algorithm for self calibration from several views. In *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR '94., 1994 IEEE Computer Society Conference on*, pages 908–912, 21-23 June 1994.

[Har97] R. Hartley. In defense of the eight-point algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(6):580–593, June 1997.

[Hir01] H. Hirschmuller. Improvements in real-time correlation-based stereo vision. In *Stereo and Multi-Baseline Vision, 2001. (SMBV 2001). Proceedings. IEEE Workshop on*, pages 141–148, 9-10 December 2001.

[Hor73] B.K.P. Horn. *The Binford-Horn linefinder*. Technical report, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, 1973.

[IHL99] Dimitrios Ioannoua, Walter Hudab, and Andrew F. Laine. Circle recognition through a 2D hough transform and radius histogramming. *Image and Vision Computing*, 16:12–26, 1999.

[Int06] Intel Corporation., 2200 Mission College Blvd., Santa Clara, CA 95052, USA. *Intel Core 2 Duo Processor*, July 2006.

[IYT92] H. Ishiguro, M. Yamamoto, and S. Tsuji. Omni-directional stereo. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 14(2):257–262, Feb. 1992.

[JD02] S. Jantz and K. L. Doty. PDA based real-time vision for a small autonomous mobile robot. In *2002 Florida Conference on Recent Advances in Robotics*, May 2002.

[KBS75] C. Kimme, D. Ballard, and J. Sklansky. Finding circles by an array of accumulators. *Commun. ACM*, 18(2):120–122, 1975.

[KDB+06] Stefan Krywult, Tobias Deutsch, Markus Bader, Gregor Novak, and Abel Gonzales Onrubia. Autonomous mirosot the autonomous way of playing mirosot. *FIRA WorlCup Dortmund*, May 2006.

[Kie92] Pär Kierkegaard. A method for detection of circular arcs based on the hough transform. *Mach. Vision Appl.*, 5(4):249–263, 1992.

[Kon97] K. Konolige. Small vision system: Hardware and implementation. In *Eighth International Symposium on Robotics Research*, 1997.

[Kry06] Stefan Krywult. *Real-time Communication Systems for Small Autonomous Robots*. Master's thesis, Vienna University of Technology, 2006.

[KYY05] Chung-Hsien Kuo, Chun-Ming Yang, and Fang-Chung Yang. Development of intelligent vision fusion based autonomous soccer robot. In *Mechatronics,*

2005. ICM '05. IEEE International Conference on, pages 124–129, 10.-12. July 2005.

[LB03] Shih-Schon Lin and R. Bajcsy. High resolution catadioptric omni-directional stereo sensor for robot vision. In *Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference on*, volume 2, pages 1694–1699vol.2, 14.-19. September 2003.

[LT87] R. Lenz and R. Tsai. Techniques for calibration of the scale factor and image center for high accuracy 3D machine vision metrology. In *Robotics and Automation. Proceedings. 1987 IEEE International Conference on*, volume 4, pages 68–75, Mar 1987.

[LW04] Shu-Xia Lu and Xi-Zhao Wang. A comparison among four SVM classification methods: LSVM, NLSVM, SSVM and NSVM. In *Machine Learning and Cybernetics, 2004. Proceedings of 2004 International Conference on*, volume 7, pages 4277–4282vol.7, 26.-29. Aug. 2004.

[MON04] S. Mahlknecht, R. Oberhammer, and G. Novak. A real-time image recognition system for tiny autonomous mobile robots. In *Real-Time and Embedded Technology and Applications Symposium, 2004. Proceedings. RTAS 2004. 10th IEEE*, pages 324–330, 25.-28. May 2004.

[NBM04] G. Novak, A. Bais, and S. Mahlknecht. Simple stereo vision system for real-time object recognition for an autonomous mobile robot. In *Computational Cybernetics, 2004. ICCC 2004. Second IEEE International Conference on*, pages 213–216, 30. August-1. September 2004.

[NH04] H. Nobuhara and K. Hirota. Color image compression/reconstruction by YUV fuzzy wavelets. In *Fuzzy Information, 2004. Processing NAFIPS '04. IEEE Annual Meeting of the*, volume 2, pages 774–779Vol.2, 27.-30. June 2004.

[NM05] G. Novak and S. Mahlknecht. TINYPHOON a tiny autonomous mobile robot. In *Industrial Electronics, 2005. ISIE 2005. Proceedings of the IEEE International Symposium on*, volume 4, pages 1533–1538, June 20-23, 2005.

[NRB+06] G. Novak, C. Roesenery, M. Bader, T. Deutsch, S. Jakubekz, S. Krywult, and M. Seyrz. The tinyphoon's control concept. In *ICM06*, 2006.

[Omn05] OmniVision Technologies, Inc. *OV7660/OV7161 CMOS VGA (640x480) CameraChip with OmniPixel Technology*, 1.91 edition, January 2005.

[PB99] Maria Petrou and Panagiota Bosdogianni. *Image Processing: The Fundamentals*. John Wiley & Sons, Inc., New York, NY, USA, 1999.

[PH98] David A. Patterson and John L. Hennessy. *Computer organization and design (2nd ed.): the hardware/software interface*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1998.

[PKG99] Marc Pollefeys, Reinhard Koch, and Luc Van Gool. Self-Calibration and Metric Reconstruction Inspite of Varying and Unknown Intrinsic Camera Parameters. *Int. J. Comput. Vision*, 32(1):7–25, 1999.

[RP66] Azriel Rosenfeld and John L. Pfaltz. Sequential operations in digital picture processing. *J. ACM*, 13(4):471–494, 1966.

[RRN02] A. Rowe, C. Rosenberg, and I. Nourbakhsh. A low cost embedded color vision system. In *Intelligent Robots and System, 2002. IEEE/RSJ International Conference on*, volume 1, pages 208–213vol.1, 30.September-5. October 2002.

[SPV05] D. Scaramuzza, S. Pagnottelli, and P. Valigi. Ball detection and predictive ball following based on a stereoscopic vision system. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 1561–1566, 18.-22. April 2005.

[SRTSB92] R. Safaee-Rad, I. Tchoukanov, K.C. Smith, and B. Benhabib. Three-dimensional location estimation of circular features for machine vision. *Robotics and Automation, IEEE Transactions on*, 8(5):624–640, Oct. 1992.

[Ste02] James Stewart. *Calculus: Early Transcendentals 5th edition - Textbook*. Number ISBN = 0534393217. Brooks Cole, 5 edition edition, December 2002.

[SWA+02] Y. Sakagami, R. Watanabe, C. Aoyama, S. Matsunaga, N. Higaki, and K. Fujimura. The intelligent asimo: system overview and integration. In *Intelligent Robots and System, 2002. IEEE/RSJ International Conference on*, volume 3, pages 2478–2483vol.3, 30. September-5. October 2002.

[SZCL00] M. Simoncelli, G. Zunino, H. I. Christensen, and K. Lange. Autonomous pool cleaning: Self localization and autonomous navigation for cleaning. *Auton. Robots*, 9(3):261–270, 2000.

[Tsa86] R.Y. Tsai. An efficient and accurate camera calibration technique for 3D machine vision. *In Proc. IEEE Computer Vision and Pattern Recognition*, 10:364–374, 1986.

[Tur94] K. Turkowski. *Fixed Point Square Root*. Technical Report 96, Advanced Technology Group Apple Computer, Inc., October 1994.

[TV98] E. Trucco and A. Verri. *Introductory Techniques for 3-D Computer Vision*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1998.

[WJ04] E.T.P. Wong and R. Jarvis. Real time obstacle detection and navigation planning for a humanoid robot in an indoor environment. In *Robotics, Automation and Mechatronics, 2004 IEEE Conference on*, volume 2, pages 693–698vol.2, 1.-3. December 2004.

[YYL05] Ying-Jie Ye, Yi-Rung Yang, and T.-H.S. Li. Full autonomous middle size soccer robot. In *Mechatronics, 2005. ICM '05. IEEE International Conference on*, pages 451–456, 10.-12. July 2005.

[Zha00] Z. Zhang. A flexible new technique for camera calibration. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(11):1330–1334, November 2000.