Robust Recognition of a Flying Ball with a Moving Camera-Inertial Sensor

Diplomarbeit

Universität Bremen, Fachbereich 3 - Mathematik und Informatik
Florian Penquitt

10. März 2008

Erstgutachter und Betreuer	Dr. Ing. Dipl. Inform. Udo Frese
Zweitgutachter	Prof. Dr. rer. hum. Biol. Kerstin Schill

Erklärung

Hiermit versichere ich, dass ich die vorliegende Diplomarbeit bis auf die offizielle Betreuung selbst und ohne fremde Hilfe angefertigt habe. Sämtliche aus anderen Veröffentlichungen wörtlich oder sinngemäß entnommene Formulierungen bzw. Abbildungen sind als solche kenntlich gemacht worden. Alle weiteren Hilfsmittel sowie verwendete Fremdarbeiten wurden ebenfalls vollständig angegeben.

Lohmar, den 10.03.2008

(Florian Penquitt)

Inhalt

Im Rahmen dieser Diplomarbeit wurde ein Hard- und Softwaresystem zur automatischen Erkennung fliegender Bälle mit Hilfe eines freibeweglichen Kamera-Inertial Sensors entwickelt. Das System ist in der Lage, einen fliegenden Ball in einem zuvor aufgezeichneten Datensatz zu erkennen und benötigt hierzu keine oder nur sehr geringe Einstellungszeit.

Die Arbeit beschäftigt sich hauptsächlich mit den Problemen und Lösungsansätzen, die sich aus der unsicheren Umgebung des Systems ergeben. Des Weiteren werden die Vorteile einer Kombination von Informationen über Bild und Flugbahn zur Erkennung des Balles untersucht, welche sich ähnlich den Vorgängen im menschlichen Gehirn gegenseitig stützen und zu einem robusten Gesamtsystem führen sollen. Die zusammengetragenen Inhalte sollen im letzten Schluss die Frage beantworten, ob bzw. wie weit ein Softwaresystem für das Fernziel 2050 des RoboCup bereits heute entwickelt werden kann. Hierbei müsste ein humanoider Roboter in der Lage sein, in einer Mannschaft mit anderen Robotern gegen den menschlichen Fussballweltmeister nach offiziellen FIFA Regeln zu gewinnen.

Die noch nicht realisierbaren Anforderungen an die Mechanik für einen humanoiden Roboter werden dabei durch einen Experimentaufbau umgangen, wobei ein menschlicher Spieler alle notwendigen Sensoren zur Datenaufzeichnung trägt und wodurch der Wahrnehmungsteil bereits jetzt unter authentischen Bedingungen untersucht werden kann.

Abstract

This diploma thesis describes a hard- and software system that is capable of detecting a flying soccer ball in a data set previously obtained with a free-moving camera-inertial sensor. The proposed system does not require the user to make a lot of adjustments, but basically works out-of-the-box under natural outdoor conditions.

The thesis focuses on the problems arising from the uncertain environment conditions, points out the benefits and flaws of the chosen approaches and ultimately tries to answer the question whether it is already possible to build at least a part of the essential software that could be used for a humanoid soccer robot competing in a 2050 RoboCup scenario today. Hereby a humanoid robot within a team of other humanoid robots would have to be able to win against the human worldchampion team according to official FIFA rules. The still too complex mechanical demands for building a robot with human-like movement capabilities are avoided by using an experiment setup where the sensors are carried by a human player.

The thesis also examines the possible benefits of a combination and optimisation of image processing data and trajectory prediction information. These information shall backup each other like it is propossed to happen within the human brain and in the end form a more robust overall system compared to a common image processing system.

Contents

1 Introduction

1.1 Overview

This document is organised into five major sections. The first section forms the *Introduction* to the general context of this work and presents its motivation. After similar systems have been briefly introduced, an outline of the proposed system and its design will be given. This outline is used to set goals and to identify the necessary preparations for achieving these goals.

The *Theory* section describes relevant algorithms from areas of computer science such as image processing, optimisation or dealing with movement in three-dimensional space. The third section presents the *Software* that was developed, shows its architecture and reviews the approaches, tools and problems during the development. The performance of other (tested but rejected) solutions will be pointed out for comparison reasons and in order to give the reader a better impression of the actual system's quality.

Afterwards, the *Results* section presents a comparison of the system's output for some selected scenes which will allow the reader to judge whether and to what extend the system fits the set demands and goals. A final conclusion and outlook of possible future work together with a statement of acknowledgements and the mandatory lists of references, figures and tables form the end of this document.

1.2 Motivation

RoboCup [1] is an international project to propel artificial intelligence, image processing, robotics and similar related fields of computer science. There are currently two different types of predefined scenarios within the RoboCup: Within different *Soccer* leagues the teams compete against each other in soccer matches that resemble the rules and settings of real soccer matches as well as possible under consideration of the technical circumstances of the league.

The *Rescue* leagues provide catastrophe scenarios in which the robots have to locate as many human victims inside an area of debris as quickly as possible. Both scenarios may either be simulated or real, forming currently 10 different leagues in total. *RoboCup@Home* is a recent addition to the first two leagues. Teams in this league demonstrate robots that are capable of dealing with everyday problems inside a human living room, e.g. finding and retrieving a specific item or operating home entertainment devices. Within all of these scenarios a very wide range of technologies can be applied and tested. The RoboCup federation's very ambitious goal is to "develop a team of fully autonomous humanoid robots that can win against the human world champion team in soccer" [1] by the year of 2050. As will soon be clarified, this goal implies much more requirements than one might think at the first glance.

Without question the mechanics that will be necessary to build a robot with actual human-like locomotion capabilities still need more development time. The RoboCup humanoid league and other robot research labs have also already shown that even simple looking tasks like bipedal walking, getting up from the floor or kicking a ball are very tough challenges for both hard- and software of the corresponding robot system. Honda's humanoid robot ASIMO [2] has been among the world's state of the art humanoid robots for a very long time now. Its latest version is already able to actually run at about 6 km/h, can move or manipulate objects and interact with humans. Still, when compared to human motions all of the currently possible actions are only at a fraction of human capabilities. Actuators are still too weak, the power sources and light-weight building materials with appropriate characteristics can be hard to find and will be very costly in the majority of cases.

Sure enough there were a lot of noticeable advancements in every RoboCup league during the past years. Field sizes were extended, the number of landmarks was reduced and the lighting conditions became more and more variable. The recognition of balls has already been researched multiple times, especially within the RoboCup were this is a very vital aspect in every soccer related league. All the systems used in RoboCup still have the very large advantage of a well specified environment. Even though these specifications are constantly relaxed they still include fairly constant lighting, predefined colours for the involved objects like the ball, orientation landmarks and most importantly the very liberal setup times of at least an entire day to configure and fine-tune the systems.

The main motivation of this thesis is to ease several of the existing vision related restrictions and to examine the efficiency of combining and optimising visual and trajectory information to a robust fused output. By having a component that allows the rating of the ball trajectory (dynamic model Z^2), another component to rate the resemblance of an object on the field to a ball and a third component to optimise these inputs, the problems that arise from easing the restrictions can hopefully be dealt with.

A common image processor would most likely recognise player heads or other round objects on the field as a ball over and over again. The proposed system would remove these preceptions because they might fit the image component but not the trajectory. By using optimisation on a segment of preceptions it would be possible to not only detect the ball (or other objects) in the current system state but to also incorporate past and even future measurements. Assuming that computation power will continue to increase a system would be able to work on a very large segment of data instead of only a single image which is evaluated by common image processing systems. Combining several sensors and bringing context into play could also reduce the error rate of a system, because it does not depend on a single information source anymore.

The main reason why a vision system with fewer configuration requirements could still work properly is the fact that its environment (field, landmarks, player characteristics) remains well defined. By easing restrictions and developing solutions for the arising problems it will hopefully be possible to give an answer to the question whether the software is already capable of dealing with the circumstances of the RoboCup goal for 2050. In order to give the reader a better impression of this task's scope we will now have a brief look at already existing systems and examine the differences between them and the system that is to be developed.

1.2.1 Similar Existing Systems

Ballcatcher Frese et Al. worked on a German Aerospace Centre demonstrator (Figure 1) [3] which was shown at the Hannover Fair 2000. A newly developed lightweight robot arm and a new type of hand autonomously caught a ball thrown towards the stationary system. The flying ball was detected with a stereo camera under the condition that there were no other moving objects (e.g. bypassing humans) in the sight of the camera. Under these conditions difference images could be used to quickly identify the current location of the ball in the images and 3D space. The extracted information could then be used to predict the ball's trajectory with an Extended Kalman Filter (EKF). As will be shown later, the proposed system also uses difference images to detect a ball to start calculations with. Before the Ballcatcher could operate properly it needed to be configured by doing several test throws. Once calibrated it was able to precisely catch the flying ball without any problems.

Hawk-Eye The Hawk-Eye [4] system (figure 2 is the currently most sophisticated commercial officiating tool used in any type of sports. It makes use of multiple stationary high-speed cameras that track the ball in realtime, and outputs television-ready animations showing a high-precision ball trajectory e.g. for game analysis or to assist the referee in difficult situations. It has already successfully been used in cricket and tennis and is es-

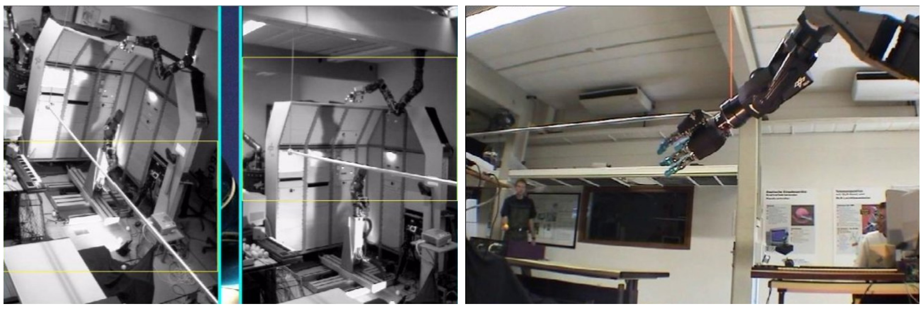


Figure 1: DLR robotic ballcatcher

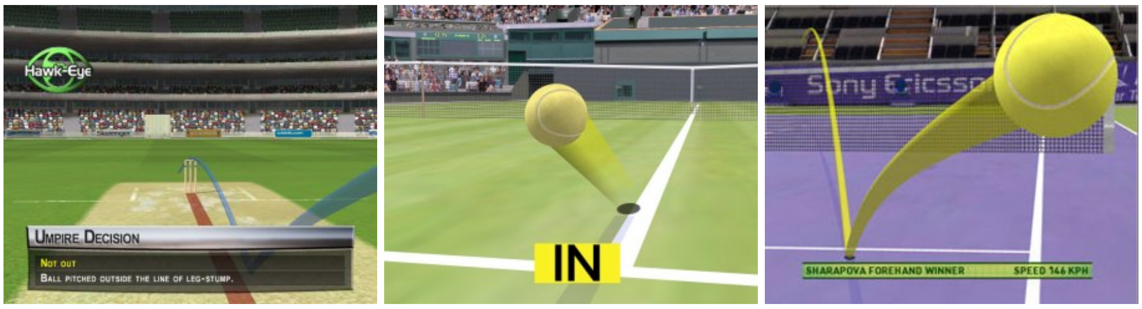
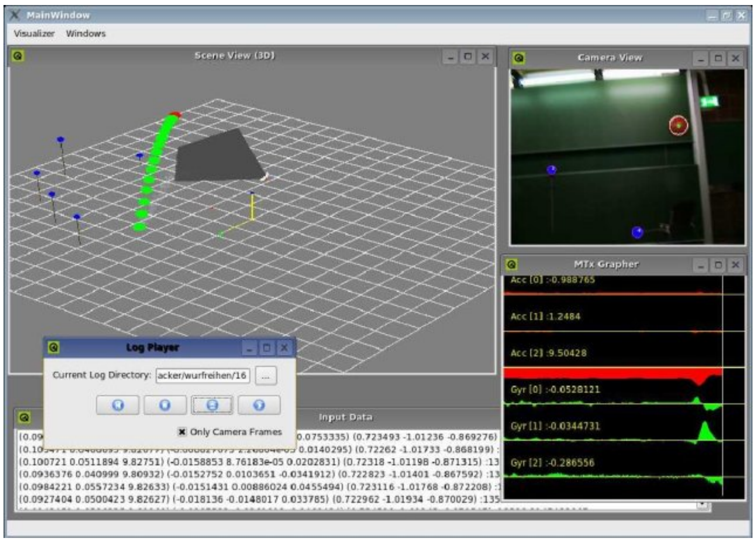

Hawk-Eye
UMPIRE DECISION
NOT OUT
BALL PITCHED OUTSIDE THE LINE OF LEG-STUMP.
IN
SHARAPOVA FOREHAND WINNER
SPEED 146 KPH

Figure 2: Hawk-Eye system

pecially interesting because of its capability to recognise balls at very high velocities under natural game settings. Neither the ball(s) nor the game field need any modifications. Still, even this professional system has problems with bright sunlight or reflections from metal surfaces. The only restriction compared to the proposed system is the stationary camera that is used to retrieve the images and the absence of secondary sensors that provide additional data. Unfortunately, it is not easy to find detailed information about the technical background of this system because of its commercial use.

Balltracker During his thesis Jörg Kurlbaum developed a system [5] (figure 3) for tracking a flying ball with a free-moving camera inertial sensor. This thesis continues his work by moving onward towards a more uncertain environment and also by emphasising the importance of camera calibration and sensor synchronisation to get more accurate results. Kurlbaum's Balltracker was already able to track an indoor flying ball and to predict its trajectory in realtime by using a Unscented Kalman Filter (UKF) [6] [7]. Since parts of his system were based on RoboCup components used by B-Smart [8], the Balltracker needed to be configured before use. Both systems use colour thresholds to segment the images in order to detect the ball and the landmarks afterwards. It will quickly have problems



```
(0.09                                                0.0753335) (0.723493 -1.01236 -0.869276)
(0.10                                                0.0140295) (0.72262 -1.01733 -0.868199) :
(0.100721 0.0511894 9.82751) (-0.0158853 8.76183e-05 0.0202831) (0.72318 -1.01198 -0.871315) :13
(0.0936376 0.040999 9.80932) (-0.0152752 0.0103651 -0.0341912) (0.722823 -1.01401 -0.867592) :13
(0.0984221 0.0557234 9.82633) (-0.0151431 0.00886024 0.0455494) (0.723116 -1.01768 -0.872208) :1
(0.0927404 0.0500423 9.82627) (-0.018136 -0.0148017 0.033785) (0.722962 -1.01934 -0.870029) :135
```

Figure 3: Jörg Kurlbaum's Balltracker

detecting the ball or landmarks when the lighting conditions change. This or a similar type of architecture is quite established and often used by RoboCup teams, e.g. the current world champion team "CMDragons" from the Carnegie Mellon University [9]. Data flows top-down from module to module whereas every module performs a certain computation (segmentation, blob construction, shape fitting, higher tactics, etc.) and passes on its output to the other modules. Therefore, every module only knows a small part of the overall information. Both the requirement of a mandatory configuration as well as the system architecture are main changes which hopefully results in a more robust and flexible system.

RoboCup Humanoids Figure 4 shows a typical game situation in the 2007's RoboCup kidsize humanoid league final match between "Team Osaka" from Japan and team "NimbRo" from Germany. The general principle of these systems fits the proposed system already quite well. The robots have free-moving cameras and autonomously track the moving ball. Using defined landmarks they can also locate themselves on the field. High-ranking teams also locate opponent players and try to avoid collisions with them. Since the rules allow additional sensors like gyroscopes or acceleration sensors, most of the teams make use of these to improve their system's performance. Even though the ball is rolling instead of flying in most cases, the scenario is quite similar to a real soccer field when not considering the scaling aspects. Yet there is a very critical difference between these systems and the proposed system. Once the light changes even a little bit, most RoboCup systems will fail to localise the ball or landmarks. This is due to the fact that almost all of

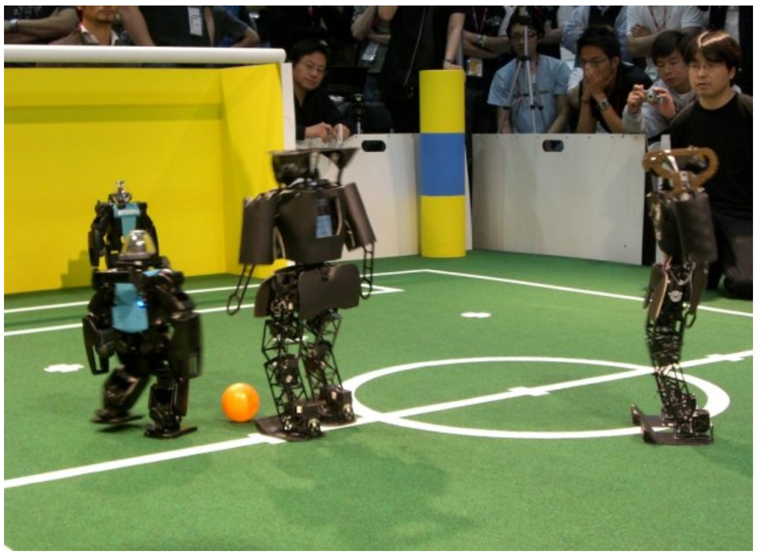


Figure 4: RoboCup Humanoids

these systems use colour lookup tables to segment the images. These tables are adjusted before every game and immediately become useless once the light changes. Of course there already are approaches that try to compensate light changes, but from personal experience it should be safe to say that most teams still rely on constant lighting in order to play at a competitive level.

Other approaches for detecting objects in the environment are of course possible and also already used, but since most platforms only carry PDAs or at most an embedded system with CPU frequencies of about 1.0 to 1.5 GHz computationally expensive algorithms on such machines may be problematic to use. Since the proposed system is currently not meant to be realtime capable and may run on a standard state of the art PC the space of possible solutions and approaches immediately increases. As it will be pointed out later, many of these possible approaches will unfortunately still not be applicable. Table 1 summaries the characteristics of the presented systems and shall give the reader a first impression of the complexity of the proposed system. The next section will then outline the proposed system more detailed.

System	Hawk-Eye	Balltracker	RoboCup	Ballcatcher	Proposed
Natural lighting	Yes	No	No	No	Yes
Predefined colours	Yes	Yes	Yes	Yes	No
Distracting objects [1]	Yes	Yes	Yes	No	Yes
Realtime	Yes	Yes	Yes	Yes	No [2]
Free-moving camera	No	Yes	Yes	No	Yes
Secondary sensors	No	Yes	Yes	No	Yes
Frames per second	≈ 200	≈ 30	≈ 30	≈ 50	≈ 54 [3]

Table 1: Overview and comparison of existing systems

1.3 Outline of the Proposed System

Overall, a special challenge is the free movement of the camera, which is directly attached to a helmet on the head of a human soccer player. This setup forms a very simple imitation of a possible humanoid robot of 2050 with human-like movement capabilities. Even though the software was originally intended to be as realtime capable as possible this is no demand of this thesis but could of course be a good follow-up topic. During my work on this thesis I quickly noticed that the high amount of uncertainty calls for algorithms that take up a lot of memory resources and computation time. The first experimental realtime version was therefore abandoned after a short period of time.

Most existing systems for recognising balls or other objects are mostly bottom-up designs [10]. After classifying pixels to certain predefined colour classes, the results will be grouped into regions (also called blobs) and then are fitted to circles or other typical shapes of involved objects that shall be detected.

From the movement of this shape a simple direction or full trajectory can be calculated later on. This approach is fast, but not very robust since the modules in most cases do not have access to the information of other modules and rely on thresholds and parameters tweaked during setup time. The pixel classificator for example will simply classify the pixels, e.g. as presented according to a colour lookup table, and does not have the information that a round object shall be recognised. Thus, the classificator will not prefer pixels that would generate round blobs if they do not fit the colour table values well enough.

[1]e.g. humans, robots, multiple balls, etc.

[2]reasons will be given later on

[3]technical maximum of the camera used

The proposed system shall follow a different approach by optimising the trajectory data combined with the image processor values. This should provide a greater robustness in situations where a far away or otherwise complicated ball could not be detected by the image processor, but can be found retroactively once the dynamic model has been established. The higher demands to computation speed and resources of the proposed system will most likely not be a problem in the future because one can assume an onholding positive growth rate in processor capabilities.

The information flow of this system is neither pure bottom-up nor pure top-down; instead the information and modules shall support each other. This approach also complies with the ideas of cognitive oriented and biology inspired image processing. Scientists have proposed that the reason for the high performance of the human visual system is the reciprocal combination of many different information levels inside the human eye and especially inside the brain instead of only relying on single local information [11] [12] [13] [14]. The proposed system shall make use of this general thought without developing a pure biology inspired system. The goals of this thesis will be defined in the following section.

1.4 Goals

This thesis uses the data and models generated in cooperation with or by Oliver Birbach [15] and its final goal is to develop a software which can recognise a standard soccer ball (and ideally also the position of a human soccer player on a real soccer field) under natural outdoor conditions. Recognising the landmarks (field points and additional marks at the field corners) proved out to be more complex than expected. Information about the position of these landmarks within the scene images is therefore received from an external log file that was also used by Oliver Birbach in his thesis.

The aim is to provide a robust system to process the data that has been previously collected in experiments without any or with only very few prior configuration. Before the actual software can be written an appropriate mount for the camera and inertial sensor needs to be constructed and all involved sensors need to be calibrated and well synchronised in order to provide the most accurate results possible.

A comparison of the proposed system and a system that only relies on image processing information shall furthermore prove the usefulness or at least the potential value of the previously outlined system's functionality.

1.5 Preparations

Before beginning with the actual work for this thesis several preparations had to be made. All of these preparations were done with great care in order to provide more accurate results later on. The preparations were carried out in collaboration with Oliver Birbach and provided the basis for his thesis, too. In order to begin constructing the hardware we were missing the following components:

- a suitable wide-angle lens for the existing camera
- a solid chessboard-like pattern for the camera calibration
- appropriate building materials (mechanics and electronics)
- appropriate tools

During the B-Smart [16] project we had contact with several companies and other university work groups or institutes that could provide us with all of the necessary components. Therefore we had everything we needed in about two weeks of time and could start the actual construction.

1.5.1 Hardware

Camera - Basler A312fc A Basler A312fc firewire camera (figure 7, left) is used to retrieve the images. The firewire port provides the power for the camera which is why no additional power source besides the laptop needs to be attached. During the experiments the laptop was either stored inside a backpack or carried by an additional assistant.

The camera can be used with a free firewire library and is running at its maximum resolution to FPS ratio which is 720x580 pixels at about 54 Hz. The images are saved in a raw Bayer format because online conversion would have taken up too much CPU resources. Attached to the camera is a Pentax H416(KP) wide angle lens with a manual aperture which allows a very wide field of view just like a human player would have. This positive feature comes along with the necessity of calibrating the camera in order to deal with the high distortion. The calibration was part of Oliver Birbach's diploma thesis [15] and allows pixel accurate detection in the output images when used together with Udo Frese's camera model software [15].

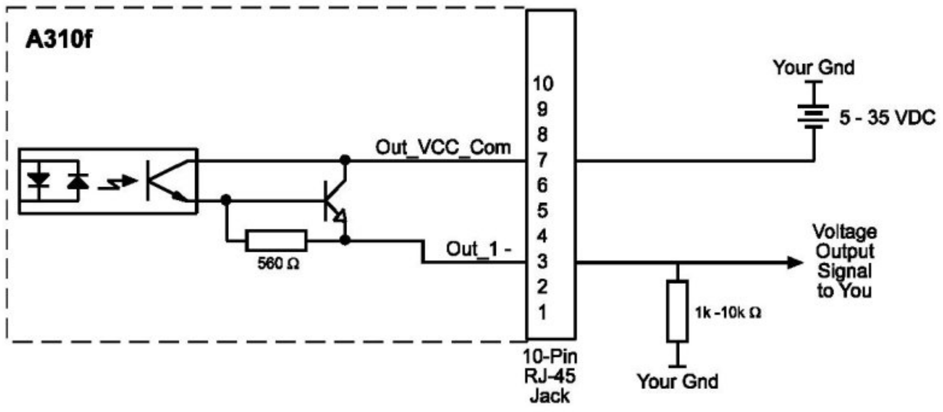


Figure 5: Schematics for camera synchronisation

The camera has a well-documented RJ50 (10P10C) port which is used for the synchronisation with the inertial sensor mounted on top of it. In order to use the synchronisation a special cable that connects the RJ50 port of the camera with the inertial sensor had to be constructed. For this purpose, the Basler manual recommended the assembly shown in figure 5 [4] which could be built and tested in very short time. With the described setup the two sensors were successfully synchronised so that the camera triggered the inertial sensor everytime an image was taken. Of course there is still a slight delay because of data transmission and storage when saving the image to the harddrive of the computer. This delay was ignored for the time being.

The MTx connectors shown in figure 6 [5] were very hard to find and even harder to solder. Since the MTx sensor is also used by other workgroup members we decided to construct the cable as an adapter that could easily be attached and removed from the sensor without actually altering the MTx hardware. The synchronisation output pin (3) on the RJ50 connector was soldered to the corresponding *syncIn* pin (3) at the MTx connector after a 1kΩ resistor was added according to the schematics in figure 5. Finally the ground pins were interconnected and the cable was carefully shielded with a heat shrink tube.

[4]taken from Basler A312f manual, section 2 - 8, figure 2 - 6

[5]taken from MTx manual, section 4 - 4 - 5

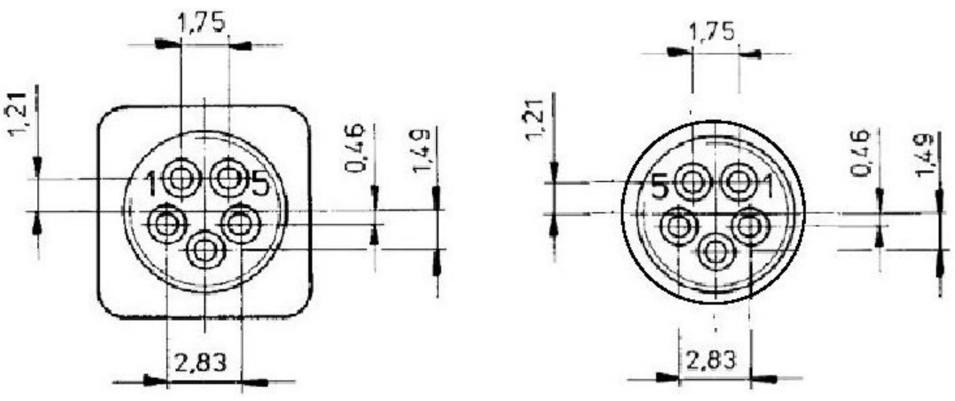


Figure 6: Schematics for inertial sensor connector

Inertial Sensor - XSens MTx The inertial sensor used is a XSens MTx (figure 7, left) model which has integrated gyroscopes, acceleration and magnetic field sensors for three axes. During the experiments the sensor proved to be very robust and also quite accurate. The hard- and software documentation which we needed to synchronise it with the Basler camera was very easy to use.

In order to equip a human player with the camera and the inertial sensor – and still enabling him to move around as freely as possible – a suitable carrying equipment was necessary. We decided that the sensors would be safe and easy to carry on a standard cycling helmet. The cycling helmet is made of reinforced styrofoam and plastic components into which we could very easily drill holes for properly mounting the camera and the MTx sensor.

We looked for an appropriate material for the sensor attachment starting off with some thin metal plates we found in the storage room. The metal could be easily bent and drilling holes was also possible. Since the sensor attachment could only be locked into position on very few points at the helmet, we looked for a more suitable material and finally decided to use Makrolon, which is a special kind of lightweight plastic. It could easily be formed when carefully heated, but remained in its formed shape once it cooled down. Drilling holes into this material was also very easy to do. The final construction was a lightweight, comfortable to wear, adjustable but still robust solution shown in figure 7.

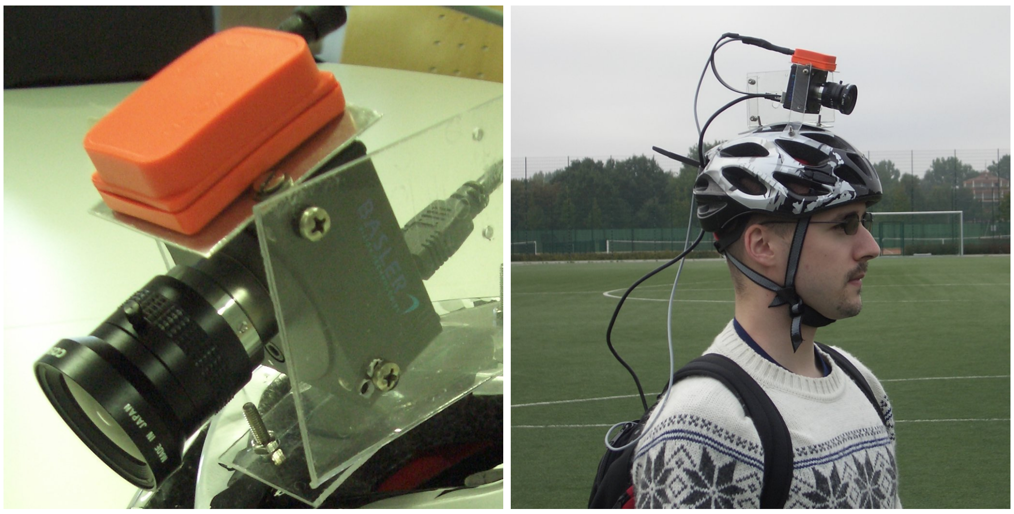


Figure 7: Inertial sensor and camera (left), worn system (right)

1.5.2 Software for Sensor Synchronisation

In order to use the hardware synchronisation feature of the camera and the inertial sensor, only the software for reading out the inertial sensor needed to be slightly modified. The camera by default already triggered the corresponding pin on the RJ50 port everytime a picture was taken. The XSens MTx inertial sensor has an integrated triggering mechanism for starting a measurement. The measurement can be activated either on a rising or a falling edge on the corresponding pin, which for our application did not seem to make any noticeable difference. Jörg Kurlbaum's *inertial server* [5] was extended to make use of this synchronisation feature by simply implementing methods for sending the packets for activating the hardware synchronisation.

1.5.3 Data Acquisition

The data (images and inertial sensor values) was recorded in cooperation with Oliver Birbach and two fellow students on a soccer field of the Universität Bremen. After the field was carefully admeasured, we recorded about 100 scenes (approximately 25GB of raw data) of typical soccer situations like freekicks, throw-ins, goal kicks or passes. An average scene has a duration of about 3 to 5 seconds, long scenes may vary between 10 to 20 seconds.

The ball is either kicked or thrown by a human player while another human player, who carries the constructed camera-inertial system, observes the trajectory of the ball while slightly moving or remaining in his starting position. Movement of the observer was done carefully because we still were not sure about how stable the entire system would be. The observer still tried to act like a *normal* soccer player in the corresponding situation as well as possible.

After the first session we noticed that data could not simply be written to the harddrive, because the laptop used – a state of the art MacBook – had an agitation sensor which deactivated the harddrive when the laptop was moved to roughly. To compensate for this problem, Jörg Kurlbaum's *camera server* [5] was also slightly changed so that images were temporarely stored in the main memory and not written to disk until the end of a scene recording. Scenes with overly large gaps were recorded again in a second session a few days later. Most scenes were taken in bright sunlight, but there are also several scenes with partly or completely clouded skies available.

The ball used in the scenes was switched frequently because we assumed that a red ball might be better for the image processing algorithms, especially at the beginning of the development phase. Since a normal soccer match is typically not played with such a ball we also used a genuine black and white soccer ball. The next section will provide some theoretical backgrounds that the reader should be familiar with, before reading the main chapters about the proposed system.

2 Theory

2.1 Rotations in 3D

Axis Angle Representation A possible way to represent a rotation in 3D space is the Axis Angle form. The rotation is represented by two values: a vector component for the direction of the rotation and an angle value which describes the rotation about the denoted axis. This way of representing a rotation is quite easy to understand. The downsides of the Axis Angle form is that it is not free of singularities and that it is not capable of transforming a point or vector in 3D space. The following simple example [6] shall demonstrate its usage:

Assuming we are standing on an even ground and choose to have the negative Z axis point just like the gravity's direction. So in other words the Z axis is pointing straight downward into the ground. We furthermore pick the positive X axis to point left and the positive Y axis to point forward from our current point of view. If we now want to denote a rotation of 45 degrees to the left (rotating about the Z axis) we could write this as the following axis angle representation:

$$\langle angle, axis \rangle = \left(\begin{bmatrix} x \\ y \\ z \end{bmatrix} \quad , \quad \theta \right) = \left(\begin{bmatrix} 0 \\ 0 \\ \frac{\pi}{4} \end{bmatrix} \quad , \quad \frac{\pi}{4} \right)$$

We can also represent this as a rotation vector pointing in Z axis direction with a magnitude of $\frac{\pi}{4}$:

$$\begin{bmatrix} 0 \\ 0 \\ \frac{\pi}{4} \end{bmatrix}$$

Quaternions Quaternions are another way of representing rotations in 3D space and can be found throughout the proposed system's software components. The reason why quaternions were used as representation form for coordinates is mostly because it was

[6]adapted from http://en.wikipedia.org/wiki/Axis_angle

	i	j	k
i	**-1**	k	-j
j	-k	**-1**	i
k	j	-i	**-1**

Table 2: Quaternion combination conditions

successfully used in Kurlbaum's thesis [5] which was the predecessor for this thesis. Mathematically seen, quaternions are simply an extension of the complex numbers defined as the ring

$$\mathbb{H} = \{a + bi + cj + dk \mid a, b, c, d \in \mathbb{R}\}$$

where each quaternion is defined as a quadruple with the form

$$q = (a, b, c, d)$$

and defined addition and multiplication operations which can be ascribed to the corresponding operations for real numbers. Quaternions can describe a rotation around three axes with four numbers and a quaternion magnitude of 1. It is important to know that multiplication of quaternions is not commutative and is subject to the combination table shown in table 2. Multiplication is done from the outer left column element to the corresponding top row element, e.g. $i \cdot j = k$. Note that all squares of the imaginary components are -1.

Multiple rotations can be achieved by simply successively multiplying (and normalising) the corresponding quaternions. A rotation by α around an axis *u* axis can be represented with the quaternion

$$q_{rotation} = (cos(\frac{\alpha}{2}), sin(\frac{\alpha}{2}), \frac{u}{|u|})$$

and a point (X, Y, Z) in 3D space may be represented with the quaternion

$$q_{pointInSpace} = (0, x, y, z)$$

The combination (rotated point) of these quaternions is defined as

$$q_{rotated} = q_{rotation} \cdot q_{point} \cdot \overline{q_{rotation}}$$

A very useful and intuitive way of writting down rotations is the form of *A2B* where *A* and **B** are two different coordinate systems, e.g. camera and world coordinates. The quaternion *camera2world* would then be a transformation of a point from camera coordinates into world coordinates. There are several good sources for more information about quaternions, beginning with the publication of the inventor of quaternions - Sir William Rowan Hamilton [17] up to modern books [18] or online resources [19].

2.2 Image Processing

This section discusses state of the start approaches for detecting circle objects in images and their application possibilities for the proposed system. It also presents a newly-developed approach for evaluating the quality of a circle and for detecting a moving ball within a sequence of images recorded with a free moving camera-inertial sensor.

2.2.1 Circle Detection

Detecting circles – or other types of shapes – in images is a very fundamental part of image processing. Eye recognition, medical applications like brain scanning, industrial applications for verifying the quality of products or the RoboCup with its demands to recognise different objects on the playing field are only a few examples for the wide range of use of circle detection algorithms.

Before working on the system's image processing component a few simple tests and inquests were made, in order to find a suitable method for detecting the flying ball within the recorded images. The following subsections each point out the positive and negative aspects of the considered methods and finally give a reason why the particular method was selected or abandoned.

Contracting Curve Algorithm The detection capabilities of Robert Hanek's contracting curve density (CCD) algorithm [20] were quite stunning when first reading the paper about this shape detection method. The algorithm was said to achieve "a high level of robustness and subpixel accuracy even in the presence of severe texture, shading, clutter, partial occlusion, and strong changes of illumination". The example images showed various shaped objects like balls, circles, cups or even humans being perfectly fitted and therefore recognised in the source images.

The CCD algorithm "applies a novel likelihood function for the assessment of a fit between the curve model and the image data which can cope with highly inhomogeneous image regions" and "uses blurred curve models as efficient means for iteratively optimising the posterior density over possible model parameters".

Even though this algorithm seemed to be very promising and interesting, it also seemed to be very complex and challenging. After a discussion with Udo Frese we decided that this algorithm would not be suited because of its difficulty and the small amount of available time to implement it. Therefore it was abandoned without any experiments at all. For future work it still might be an interesting alternative to look further into.

Robust Circle Detection using a Weighted MSE Estimator Prof. Dr. Guido Schuster and Aggelos K. Katsaggelos presented a new way of detecting circles with a weighted minimum mean square error formulation in 2003 [21]. This method also seemed to be very interesting for usage in the proposed system because it claimed to achieve "its robustness by operating in one step, using all pixels of the image (correctly weighted) and not using any thresholds" unlike the traditional approaches which in general always consist of two stages. In that first stage the source image is usually input into an edge detector (e.g. Sobel or Canny) which of course is "sensitive to noise". The second stage – which is the actual circle detector – now works on the reduced information generated in the first stage and cannot correct any errors made in the previous stage. Working without thresholds sounded promising because of the natural lighting conditions the proposed system should be able to deal with. The method furthermore claimed to consume fewer resources than the standard Hough transformation, which of course was yet another positive point.

But while looking deeper into this approach it more and more became obvious that it would not be a suitable method for detecting the ball in the scene images. The first fundamental assumption of the MSE circle detector is "that there is only one circle in the image" which for the recorded scenes of course is not true. There is currently at least on other human player on the field, possible artifacts in the surrounding fence and shadows from player heads and the ball on the ground. But at the time being this method was further investi-

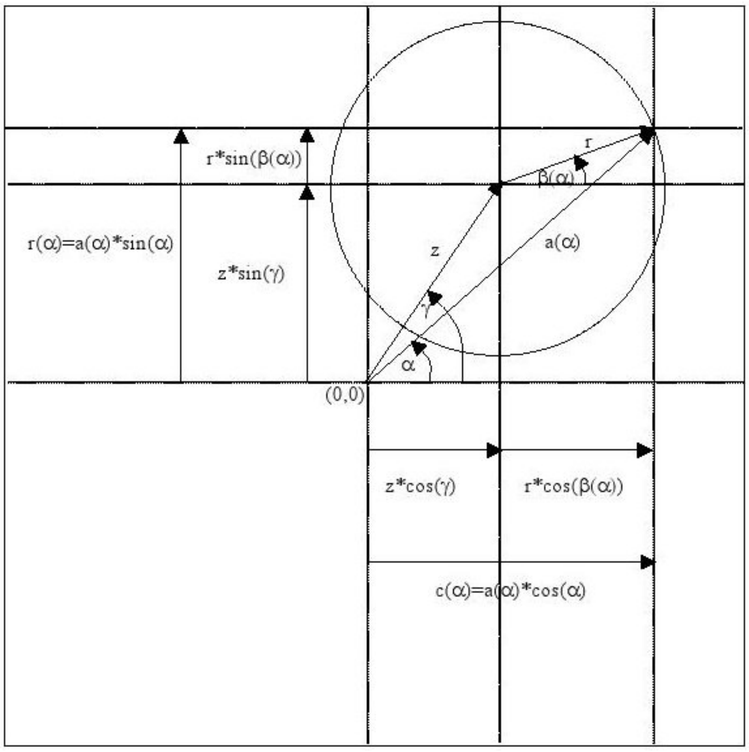


Figure 8: Circle model of MSE approach

gated anyway, under the assumption that this problem could somehow be coped with. In order to give the reader at least a slight impression of the functionality of the method – even though it was not used for the proposed system later on – the following paragraphs shall explain the approach. All of this information has been adopted from the listed paper.

With the circle model illustrated in figure 8 of the MSE method the circle's centre and radius can be calculated if the pixels, that belong to the circle, are known. The MSE circle model assumes a ray at an angle α ranging from 0 to 360 degrees originating at the image centre and covering the entire image. This circle is defined with the centre at the point of (C_x, C_y) where

$$C_x = r \cdot sin(\gamma), C_y = r \cdot cos(\gamma))$$

and a radius *r* with the length of the ray which is tracing the circle. The model furthermore assumes that $a(\alpha) \cdot cos(\alpha)$ and/or $a(\alpha) \cdot sin(\alpha)$ can be measured within pixel accuracy and therefore the circle parameters C_x, C_y and *r* can be calculated from the measurements.

To calculate the horizontal centre of the circle, the following holds:

$$a(\alpha) \cdot cos(\alpha) = C_x + r \cdot cos(\beta(\alpha))$$

with $\beta(\alpha)$ depending on α, but C_y being independent of α. The $r \cdot cos(\beta(\alpha))$ term may now be removed by taking the average over all pixels that belong to the circle. This operation is indicated by a line above the expression in the following.

Since for every pixel on the circle there is a pixel diametrically opposite to it where the $cos(\beta(\alpha))$ term has the same magnitude but opposite sign, the terms already listedabove hold for the horizontal dimension and in similar way for the vertical dimension. The radius of the circle can be calculated with the average distance between the estimated circle centre and all circle pixels. Even though this approach is faster than a Hough transformation, it requires that the pixels belonging to the circle can be easily detected, which is not that easy in noisy images. The scene images also have a much higher resolution than the example images shown in the paper pixels which will increase computation time again.

To be able to deal with gaussian-noised images the MSE approach does not actually detect the pixels that belong to the circle, but instead uses the weighted values of all pixels to estimate the circle characteristics. The approach uses the well known concept of MSE estimation via a the Moore-Penrose (generalised) inverse [22] of a matrix. Details about this can be found in [21] but have been omitted since they are of no importance to this thesis.

After a few first tests and consultation with the method's creator Schuster, it quickly became obvious that this method will have problems working with realistic images that contain more than one circle object (ball, heads of other players, parts of the fence, etc.) and have non-gaussian noise affecting the circles. It therefore was abandoned and not further tested.

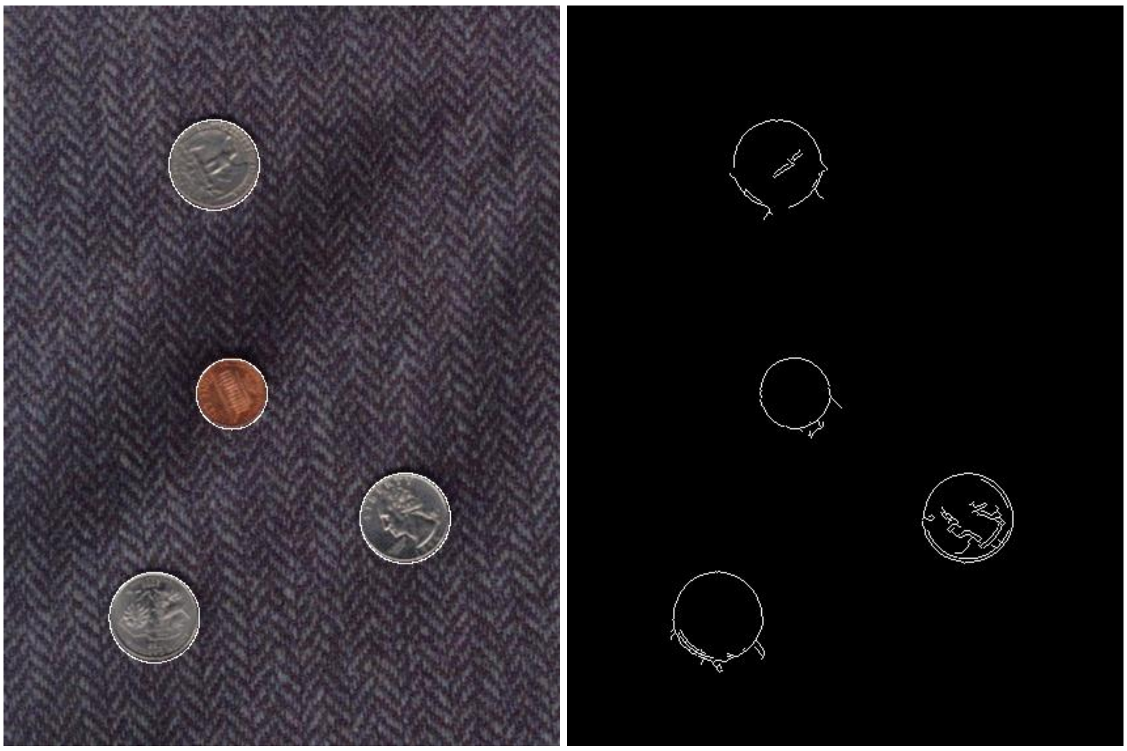


Figure 9: Source image (left), Standard Sobel image (right)

Hough Transformation The Hough transformation was originally developed by Paul V.C. Hough in 1962 [23] in order to detect lines in images. After Richard Duda and Peter Hart generalised the method in 1972, Dana H. Ballard finally popularised this method [24] which it is still used today. Since 1981 several possible improvements and modifications have been published [25], e.g. the Fast Hough Transformation [26].

Since this method was part of several lectures at the Universität Bremen, is widespread and not too difficult to implement it was considered for the proposed system's circle detector. The principle of this method shall be explained according to a well known example [7] in which coins are to be automatically detected.

Figure 9 shows the original raw image with the already detected coins (white circles) and the first step for detecting these, a Sobel filtered image with only the edges of the coins remaining, on the right. The Sobel image is generated with a standard method not further discussed.

[7]taken from http://www.math.tau.ac.il/ turkel/notes/hough4.html

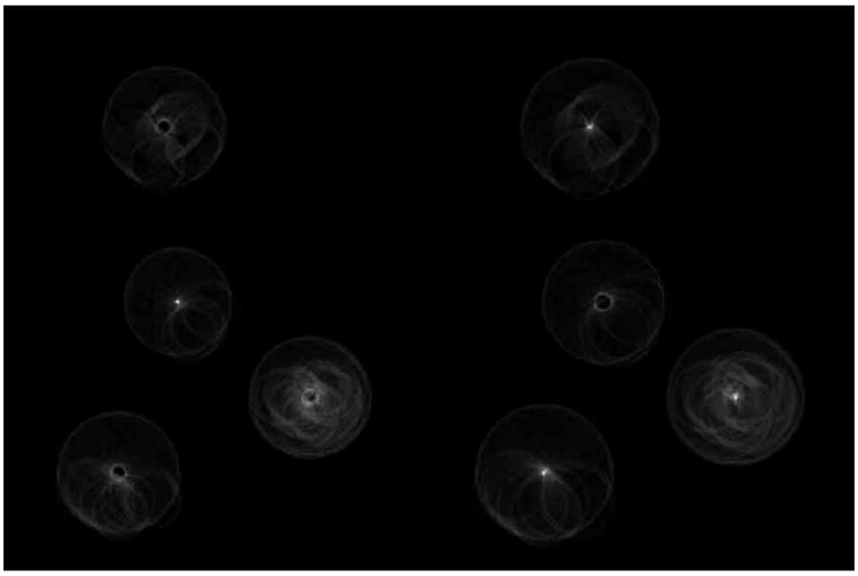


Figure 10: Coins Hough space (Pennies left, Quarters right)

After the edge image has been generated the Hough space may be generated. Circles can be represented with the term

$$(x - C_x)^2 + (y - C_y)^2 = r^2$$

The Hough space is formed by all possible circle parameter (C_x, C_y, r) combinations. When visualised like in figure 10, every pixel of the Hough picture represents one specific combination of parameters. The brighter that pixel is, the bigger the evidence for a circle with exactly these parameters would be. The Hough image's pixel values are generated by accumulating the edge values from the source image. For every set pixel in the edge image each Hough space pixel whose circle lines would run through that pixel is increased either by a constant (depends on threshold) or by the corresponding edge value (stronger edges are rated stronger). The radius is set to a specific minimum and maximum value when doing the Hough transformation. The best circles in the image can later be found by simply searching for the maxima in the Hough space. In any case, the time and memory consumption of the method is high and will increase with the image resolution and radius range.

As will be shown in the *Software* section later, the OpenCV [27] library's circle Hough transformation was not very satisfying. That is why yet another circle detection method had to be examined.

Radial and/or Tangential Error Values Even though the proposed system needs a method to actually find circles when looking for a starting ball to work with, it also needed a method to rate a potential circle's shape and colour at a specified location. The algorithm shown in figure 11 and several slight modifications of it were integrated into the system for rating the shape of a circle.

```
RATECIRCLE($x, y, r$ : double , $sobelImg$ : SobelImage )
minContrast = 2.0;
mCT = minContrast/sqrt(2.0);

for (alpha = 0; alpha < 360; alpha++)
do
    xp = floor(0.5 + x + cos(alpha) * r);
    yp = floor(0.5 + y + sin(alpha) * r);

    sobelX = sobelImg[xp, yp].x;
    sobelY = sobelImg[xp, yp].y;

    sobelLen = sqrt(sobelX * sobelX + sobelY * sobelY));
    sobelT = fabs(-sin(alpha) * sobelX + cos(alpha) * sobelY);
    Error = Error + ((sobelT + mCT)/(minContrast + sobelLen));
    PixelsAnalysed++;

mean = -Error/PixelsAnalysed;
return (mean);
```

Figure 11: Circle rating algorithm

The algorithm accesses a previously generated Sobel image. This image has the same dimensions as the source image it was generated from. The Sobel image can be interpreted as an "edge image" in which every pixel's brightness – which corresponds to the magnitude of the Sobel vector – indicates how strong each source pixel may be considered an edge inside the image. In other words the Sobel value is the gradient of the image brightness as a vector in X and Y direction. This vector is orthogonal to the edge.

Given the circle's centre at x and y and with a radius of r, the circle points can easily be calculated by using sinus and cosinus functions with an angle from 0 to (a full circle) 360 degrees (6)(7). Every circle point's Sobel length as the square root of the squared X and Y Sobel values is calculated in (12). This value simply indicates the strength of the contrast in the direction of the edge (radial).

The SobelT value computed in (13) indicates the quality of the edge in tangential direction of the assumed circle. If this pixel is really part of a circle edge this value needs to be small because the circle's border pixels would be similar. If the value is high, it would indicate a break or inhomogeneity within the circle's outer border and therefore raise its error value.

The final value for this pixel (14) is accumulated to the overall error for this circle and is the quotient of the previously computed radial and tangetial values modified by constant values for a minimum contrast for the circle edges. The basic idea behind this quotient is that the error value will increase if the radial contrast decreases and if the tangetial contrast increases. The result of the algorithm is an error value for the entire circle (17), which is the mean of the accumulated values previously described. It is returned as a negative value so that it may be used as a *believe* value when substracted from 1. This value then indicates the likelihood that there really is a circle with radius r at the passed coordinates x and y.

2.2.2 Difference Images

A difference image is an image which shows the numeric difference between the pixel values of two source pictures. The higher the value of a pixel in the resulting difference image, the more this pixel differed in the two source images. If the camera is supposed to remain at the same location the difference of two consecutive images could be interpreted as some kind of movement that happened during the time these two images were recorded.

When using a stationary camera a difference image of two images with the same width and height is very easy to create. Since every pixel of the first image corresponds to the pixel with exactly the same coordinates in the second image the difference can be calculated using the algorithm in figure 12.

Since the proposed system uses a free-moving camera, the difference images cannot be generated with this simple algorithm. In order to do this one needs camera calibration information, which were a part of Oliver Birbach's thesis [15] and a camera model, which was implemented by Udo Frese [28]. The calculated calibration data is passed to the camera calibration software on startup and calculates all necessary parameters for the actual

```
DIFFERENCEIMAGE($img1$ : sourceImage , $img2$ : sourceImage )
for $y \in height$
do
    for $x \in width$
    do
        $differenceImage[y][x] = abs(img1[y][x] - img2[y][x]);$
```

Figure 12: Difference image algorithm for stationary camera

camera (affected by the wide angle lens) used. Afterwards it is possible to deal with even highly distorted images provided by the camera and to map 3D objects within the environment to a 2D image and vice versa. The algorithm ignores the translation on the field that happened between two images for the time being, since the relative movement which can be calculated from the gyroscope values is sufficient for the current state of development. When considering the translation between the images one would also need information about the camera's location on the field which is currently not tracked yet.

Due to this circumstances the difference image generation algorithm must be extended in a way shown in figure 15 to calculate at which position every pixel in the second image was previously in the first. A 2D Pixel in the old camera coordinate system is transformed into a 3D ray in world coordinates (6). As mentioned the translation is not used, so a fixed distance of one meter is used for the pixel's distance. In the next step the relative movement of the camera is used to calculate a new transformation which is applied to the camera model (7)(8). We now can use the new transformation to transform the world point back to a 2D pixel (9). This pixel is the corresponding pixel in the second image and we can now calculate the difference between these two pixels. When calculating this value a contrast normalisation is performed first in order to reduce noise in the resulting difference image. If the pixel in the second image shows something that was not part of the first image the difference image for this pixel will simply be blank.

Figures 13 and 14 show sample difference images from an actual scene computed with the proposed system. While the second example in 14 only has one valid blob resulting from the moving ball in the original image the first example shows a lot of set pixels in an area where no actual movement took place. How these blobs are created, merged and filtered will be described in the following section.

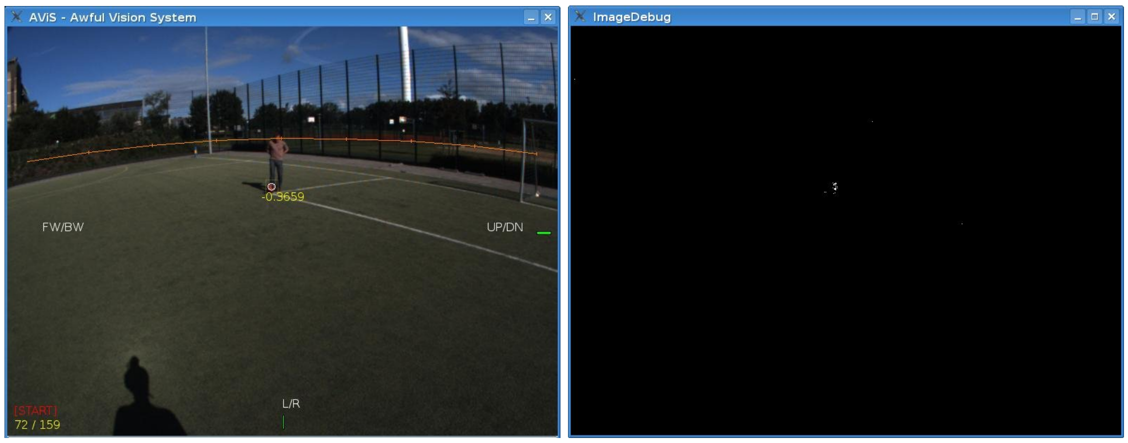


Figure 13: Difference Image with ball blob

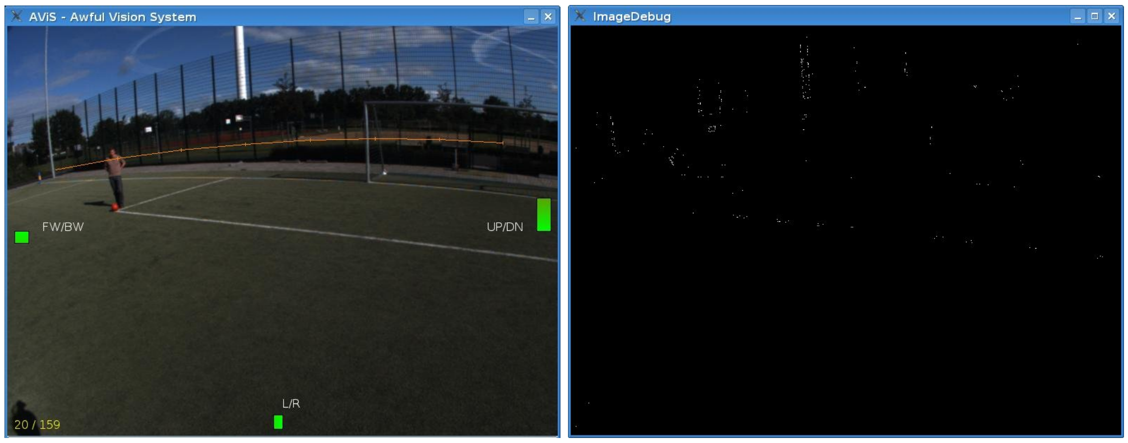


Figure 14: Difference Image with a lot of invalid movement

2.2.3 Blob Building

The term *blob* is well known in almost every RoboCup related software system which has an image processing component. The *traditional* blob in these systems is a shape that consists of connected or at least adjoining pixels that were previously segmented to be of the same colour class. Typically objects that are made of one or multiple blobs are: ball, landmarks, goals or opponent robots.

Detecting ball movement – in order to have a starting ball for the calculations – is done by grouping several pixels together which have changed between two consecutive images. This is done by using the already described difference images. To be considered a valid blob

```
DifferenceImage($img1, img2$ : sourceImage , $relativeRotationData$ : inertialData )
for ($y = 0; y < img1.height; y++$)
do
    for $x = 0; x < img1.width; x++$
    do
        $setSensor2World(cameraIsWorld)$
        $3DRay = image2SensorRay(img1.x, img1.y)$
        $newCamera2world = calculateNewCamera2world(relativeRotationData)$
        $setSensor2World(newCamera2world)$
        $2DPixel = world2Image(3DRay, newX, newY)$
        if ($hasValidLimits(newX, newY)$
          then
              $differenceImage[y][x] =$
              $normalizedContrastDiff(img1[y][x] - img2[newY][newX])$
          else
              $differenceImage[y][x] = 0$
```

Figure 15: Difference image algorithm for free-moving camera

from two consecutive images, the potential blob must fulfil several conditions, which differ from the wide-spread conditions of most RoboCup systems. Table 3 shows an overview of the conditions.

In comparison to most – if not all – RoboCup systems, a blob with the proposed system may consist of multiple pixels which are not directly connected to each other but have unset pixels throughout the entire blob. A very superficial definition of a blob within the proposed system could be that a blob is just a cluster of nearby pixels.

This tradeoff needed to be introduced because even with very strong movement between two images there are still pixels inside the ball that do not change significantly enough. Figure 2.2.3 illustrates the sequence of how a blob is actually built:

The image is scanned pixel by pixel beginning in the top left corner of the difference image. Once a set pixel has been found, this pixel will either be added to an already existing blob

	Common RoboCup systems	Proposed system
Blob has minimum size	Yes	Yes
Blob has maximum size	Maybe	No
Blob pixels are directly connected	Yes	No
Blob has a predefined shape	Yes	Yes

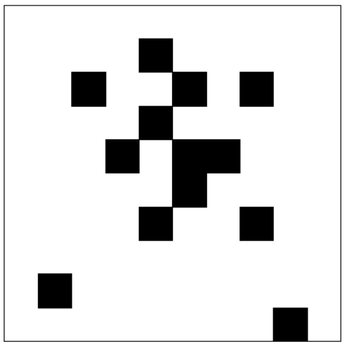
Table 3: Blobs in RoboCup and proposed system

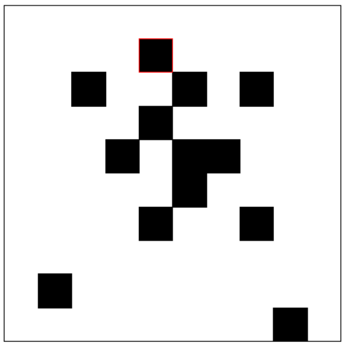
within its vicinity or to a new blob that is created at that time. When a pixel is added to a blob – as step 2 illustrates – the new borders of the resulting blob are the lowest and outmost X and Y coordinates of the pixels the blob consists of. Even though this approach might seem not very sophisticated, it worked out quite well.

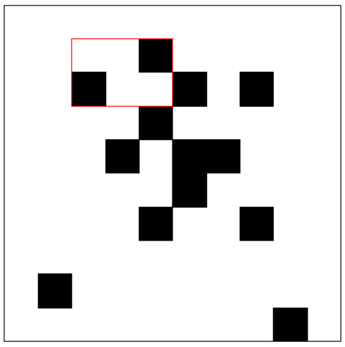
Unfortunately, with this procedure a lot of false blobs will be detected in the difference images, especially inside the fence that surrounded the soccer field on which the scenes were recorded. To get rid of these blobs, which definitely have nothing to do with a moving ball, the previously mentioned procedure was extended by merging nearby blobs together to a single bigger blob. Figure 16 shows the sequence of two blobs being merged together.

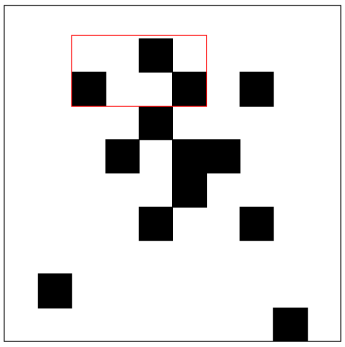
By merging the blobs inside the fence – or elsewhere on the field – a very long blob will be created which resembles the form of a slender column. Figure 13 shows a resulting difference image from the pictured source image and its predecessor. Since this column's height is by far bigger than its width the rectangular blob will be deleted at the end of the blob building sequence because it does not fulfil the last condition for a blob, which demands that all valid blobs are of quadratical shape at a maximum width to height ratio of 1.5 to 1.0 and vice versa.

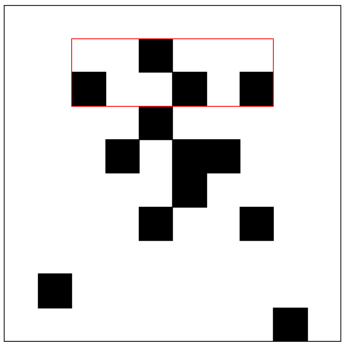
Even though this approach is not very complex, it still worked out quite well in the tests as the following sections will show. The next section will introduce the reader to the optimisation method used in the proposed system.

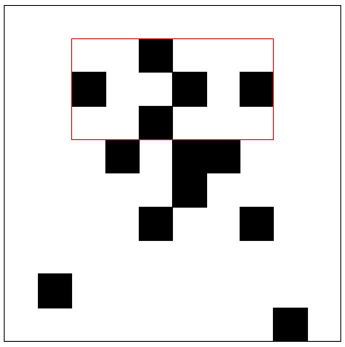

Step 1


Step 2


Step 3


Step 4


Step 5


Step 6

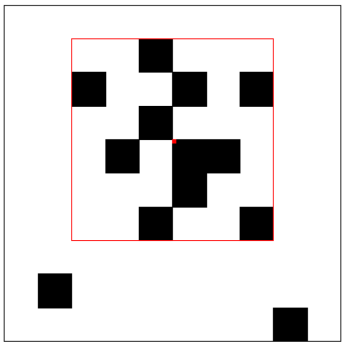

Result

Table 4: Blob building sequence

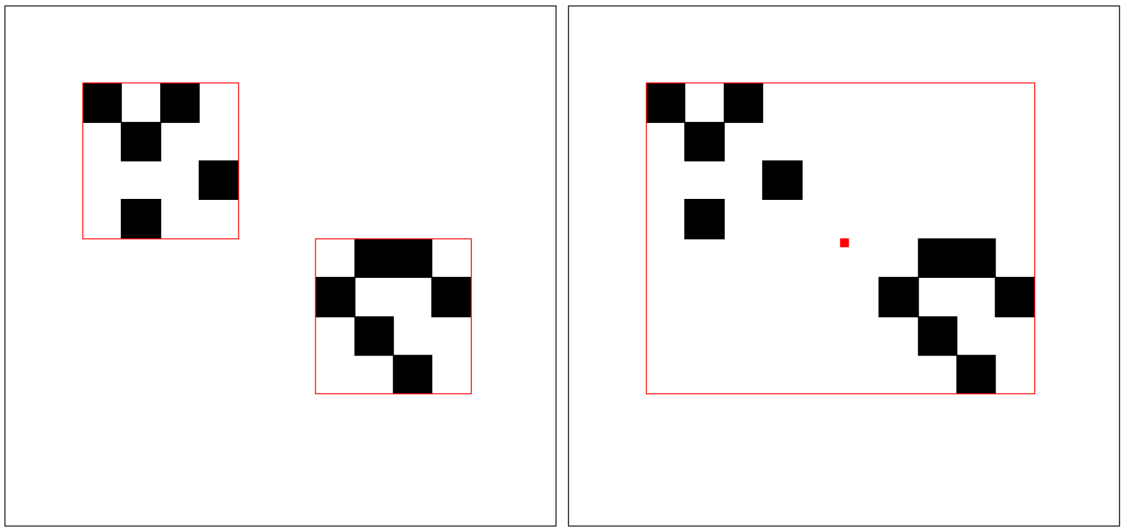


Figure 16: Blob merging procedure

2.3 Gradient Descent Optimiser

The gradient descent [29] is a standard algorithm for general purpose function optimisation. The algorithm finds a local minimum of the function $F(x_{last})$ by continuously taking steps in the direction of the negative gradient $-\delta F(x_{last})$ at the last evaluated point x_{last}, finally converging in a functions's local minimum. A step width factor ϵ is used for the actual range of the step since the gradient only indicates the direction of the step but not the length. A constant a calculated value for ϵ may be used where the constant value is of course easier to implement but a dynamic step width will most likely result in better performance.

There are two variations of the algorithm when implementing it: the usage of analytical or numerical derivation. The first alternative requires that the function that shall be optimised can be derived arbitrarily. This is possible for trivial functions but very difficult for the error calculation function used in the proposed system. This function is complex and its precise characteristics are uncertain. The error calculation in the proposed system depends on 16 values and has dependencies that are hard to determine.

Hence the second alternative – numerical derivation – was used for the implementation. Hereby the gradient of the function is calculated by evaluation the function at $F(x)$ and slightly altering the n-th component of the input vector by ϵ either up or down. By using this method one does not need to construct or even know the first derivation of the function that is to be optimised.

```
BLOB-BUILDING-ALGORITHM($DifferenceImages$)
 1  for $y \in height$
 2  do
 3      for $x \in width$
 4      do
 5          if $pixelIsSet(y, x)$
 6            then
 7                if $!blobIsNear(y, x)$
 8                  then
 9                      $createNewBlobAt(y, x);$
10
11                $addPixelToBlobNear(y, x);$
```

Figure 17: Blob building sequence

But even though the algorithm is known to finally converge into a local minimum of the function there are several problems especially when considering the use in the proposed system. The algorithm will quickly descent into the right direction of the minimum of $F(x)$ but final convergence will take up a lot of time. It will take even more time when the algorithm is started with poor starting values. Guessing these values for a complex function as it is used in the proposed system is not easy. The solution for this problem is described in the *Implementation* section.

3 Software

3.1 General Architecture

Figure 18 shows an illustration of the general architecture of the proposed system. The scene data, which in the current state of development consists of the visual image data, the inertial sensor information and the locations of landmarks in the individual images, is loaded into appropriate storage containers at the start and can be accessed from the optimiser and each evaluator.

The GUI component is used for visualising the scene data in its latest state. A sub-component of the *Optimiser* - the *Motion Detector* - is able to find a suitable start ball for the optimisation process. The *Motion Detector* detects balls in an adjustable amount of consecutive frames and updates the scene data accordingly. In the next step, the optimiser is able to optimise the scene frames by using the initial balls found by the *Motion Detector* and predicted balls in the following frames.

All evaluators receive the ball position or other important scene data converted into a 2D perspective position since the work on 2D images. They can therefore evaluate the quality of potential balls at the corresponding locations and indicate how "*ball-ish*" the target zone looks. All evaluators return error values which can be used as a part of the entire scene error. The second part from the scene error is the result of the dynamic Z^2 function which outputs an error value for the ball trajectory.

In the current state of development there are two evaluators: one for the ball's colour and one for its shape. Additional evaluators can be easily be added by using the evaluator base class and overloading the evaluation method with custom user code. The evaluators' error values will be summed up and are combined with the trajectory error values. This input is used within the optimiser during each step of optimisation until the combined error has reached its minimum.

The following *Implementation* section describes the details of each component from a more technical point of view.

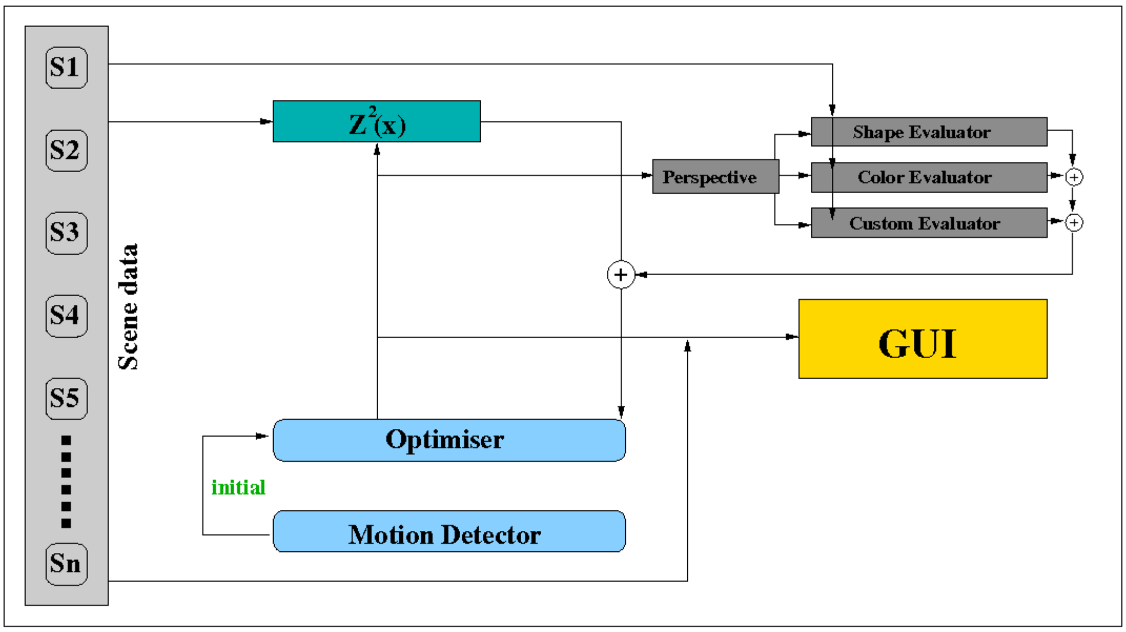


Figure 18: Proposed system's architecture

3.2 Implementation

The entire software was written in C++. Most of the described parts, as for example the optimiser or the Kalman filter, have been implemented as independent classes, which have access to the separated GUI components. A mathematical framework by Udo Frese has been used extensively throughout the entire system in order to work with matrices, vectors, transformations and camera calibration.

3.2.1 Optimiser

The optimiser component currently uses the already described Gradient Descent algorithm to optimise the trajectory data. The function that is optimised outputs the total error value for the set segment of trajectory data and is a combination of trajectory error and image processing errors. In contrast to Oliver Birbach's thesis the location of the ball was not measured before and input into the system. Instead this (image processing) part was replaced with another real error value, which indicates how "*ball-ish*" the area in the image where the trajectory assumes the ball actually is. Hence by reducing this output value the quality of the ball recognition will increase. The optimiser currently works on a flat vector

Vector index	Data
01-03	Ball position in world-coordinates (X, Y, Z)
04-06	Ball velocity in world-coordinates (X, Y, Z)
07-09	Camera position in world-coordinates (X, Y, Z)
10-12	Camera velocity in world-coordinates (X, Y, Z)
13-16	Rotation (inertial2camera) in quaternion form

Table 5: Optimiser vector

(shown in figure 5 consisting of 16 values:

This vector contains the parameter values such as the balls position that is forwarded into the dynamic model functions from Oliver Birbach's thesis [15] and is also used by the evaluators described later on. Before the values can be used, they have to be converted into a 2D perspective which is done by using existing transformation code from the camera model class. The evaluators now calculate their own fraction of the frame's error just like the dynamic model functions do. The overall frame error is the sum of all evaluator errors and dynamic model errors. Summing up all total frame errors results in the overall scene error that the optimiser is working on and which we want to reduce to a minimum.

After working on the flat vectors, the scene data is written back to the high level scene data containers, so that the data may be visualised by the GUI component. The optimiser implemented in the proposed system uses a dynamical scaling of the step width factor ϵ. The scaling is not very sophisticated and just follows two simple rules:

1. If the error is currently being reduced, which means we are going into the right direction, we might even go faster into this direction and therefore increase the step width by using the old value multiplied with an adjustable constant.
2. If the error is not reduced, which means we have gone into the wrong direction we reduce the step width by scaling it down with an adjustable value. We now hope that we have not gone too far into the wrong direction and by scaling down the step width will soon more into the right direction again.

As mentioned in the *Theory* chapter the gradient descent algorithm only converges into a **local** minimum and needs plenty of time to do so. Since the error calculation function does not necessarily have only one minimum this algorithm is actually not very suited for the proposed system, especially when considering the fact that it will take an excessive amount of time when the algorithm is not started up with some appropriate values.

But since the system should only show that the general idea behind all this (combining vision and trajectory data to an optimised, more robust result) is worth exploring, this method was used because it is known to be easy to implement. As the later *Tests* will show it definitely was not the best choice, but at least provided results in time.

3.2.2 Evaluators

Colour Evaluator The *Colour Evaluator* rates the resemblance of a circle with given values to either a colour or black and white ball. When rating the resemblance to a colour ball the evaluator must be told which average colour (RGB value) the real ball will have. Unfortunately, the evaluator will also need thresholds for the RGB values in order to decide whether a circle's pixel matches the real ball or not. Since the ball is affected by lighting changes while flying, evaluating the colour can quickly become complicated.

Rating a black and white ball is done by just counting circle's pixels that are within a predefined range, either close to black or close to white. For both colour evaluations the ball matching pixels compared to the entire amount of pixels the ball consists of will form the returned value. Since there might be other objects on the field which resemble the balls colour using only this evaluator would be inefficient and most surely produce bad results.

Shape Evaluator The *Shape Evaluator* is an exact implementation of the algorithm already described in the *Theory* section before. During the development phase the core algorithm for evaluating a circle was tested with slight modifications, e.g. either using the tangential or radial error values, using a mean of all pixel errors or weighting the errors differently. A small test application was developed to test the efficiency of four different circle algorithm versions. Two versions did not rate the real ball as best circle in the image while the other two had noticeable time differences. The currently used version was chosen because it was noticeably faster while rating the real ball as the best circle in the image.

More Custom Evaluators A base for future evaluators is provided as part of the software system. The evaluator method must be provided with an image to work on and has access to other important scene data it might want to use. The user should thereby be able to quickly integrate more evaluators into the system.

3.2.3 Unscented Kalman Filter

The Unscented Kalman Filter (UKF) was ported from a filter originally written in JAVA, which was part of Udo Frese's "Theorie der Sensorfusion" lecture. The UKF is currently only used for filtering the camera-inertial system's rotation that will be used when displaying the artificial horizon. The filter works just like the UKF that was the main part of Kurlbaum's thesis in 2007 [5] and Oliver Birbach's thesis in 2008 [15].

The artificial horizon is a useful hint for the human user but is also used when looking for start balls. Balls above the artificial horizon will thereby be ignored and are not considered as valid start balls.

An artificial horizon can be created by building a complete 360 degrees circle of points around the observer with a specified distance between the points, e.g. 10 degrees. The location in world coordinates of this point can be calculated by using trigonometric functions with the running degree value. If this world point can currently be seen in the image, the point is added to an array of points that will finally be interconnected by a simple line. Figure 19 illustrates the algorithm for creating the artificial horizon.

At an early state of development a *Statistics Evaluator* was meant to be implemented, too. This evaluator should have previously recorded (learned) information about the characteristics of the involved balls and then compare these to the balls in the recorded images. This evaluator was left out in agreement with the tutor because it would have used up too much of the remaining time.

3.2.4 Motion Detector

Based on the presented blob building method a flying ball may be recognised. Since the ball is moving in the image sequence once it is kicked, it will generate movement blobs after entering a certain perimeter (about 15 meters) around the observer. A compromise was made that only scenes would be used where the ball's starting position was directly on the ground. By making this assumption false movement blobs – e.g. inside the fence – could be eliminated because they were above the calculated horizon.

```
DRAWHORIZON($currentRotation$ : Quaternion )
inertial2world.setRotation(currentRotation);
world2camera = inertial2world * camera2inertial;
for $point \in horizonPoints$
do
    $degrees = (10 * point) * (PI/180);$
    $nextHorizonPoint = (45 * cos(degrees), 45 * sin(degrees), 0);$
    $cameraModel.world2Image(nextHorizonPoint, x, y);$
    if $(validImageBoundaries(x) AND validImageBoundaries(y))$
      then $pointsToDraw.insert(nextHorizonPoint);$

$drawLineFromPoints(pointsToDraw);$
```

Figure 19: Artificial horizon algorithm

A big disadvantage of this constraint is that it leads to late ball recognition if the ball is kicked from greater distance. Once the movement is detectable the ball might already have risen above the horizon. It will be considered a valid start ball once it falls below the horizon which unfortunately might be at a late phase of the trajectory. In other scenes where the ball is away about 10 meters the starting ball is detected about 1 to 2 frames after it leaves the ground.

In order to be considered as a starting ball the following conditions must be fullfilled:

1. There must have been movement (blobs) in a sourrounding area over a specified number of frames (currently 3)
2. Area must be below the horizon

The *Motion Detector* was able to detect correct start balls in most of the scenes (shown in figure 20, even though these balls sometimes may be not at the beginning of the ball trajectory. Due to sunlight effects on the ball, the detection is sometimes not exactly accurate as shown in figure 21. The number of balls to detect can be freely adjusted and is currently set to 2 balls. Ball positions for the frames will be calculated from the previous ball position and its speed.

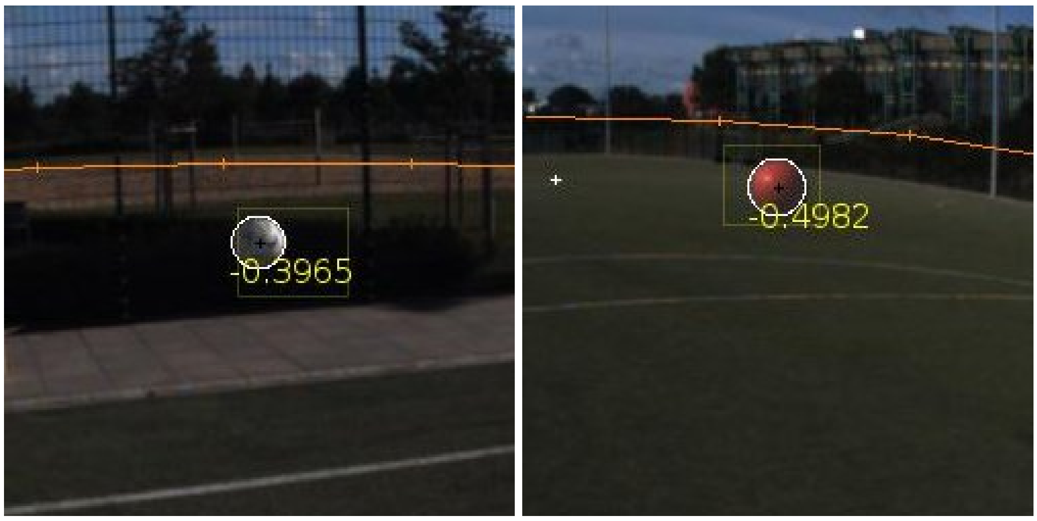


Figure 20: Correctly detected start balls with displayed (negative) error values

3.2.5 GUI

The GUI was written using the free Trolltech Qt library [30] in version 4.2. Even though the Qt library has a very comprehensive amount of classes for almost every purpose and it is quite easy to generate good results in only a short amount of time, it proved out to be not the best choice for the proposed system during the implementation.

Drawing functions are event based. So in order to draw within a Qt widget the user must overload a protected method with his own code. This means, that the component that shall be redrawn needs to know the latest state of the involved information to draw at all time. The scene display has extensive *update* methods which set the information to be visualised to the passed values. These methods may then be called by other components, e.g. the optimiser or the evaluators. The big disadvantage of this event based drawing is that other components cannot just draw debug shapes into the scene images, which can be very useful during the development of a new evaluator or while debugging it. Having the system read in all necessary data from a command line argument, processing it and writing the results consisting of images with output overlays and some logfile back to the harddrive would have been sufficient. The output images could then be displayed with the users image viewer of choice or be converted into a standard AVI video e.g. using the freely available MEncoder [31].

Other image processing related parts of the system were developed with the OpenCV library [27]. In the current state of development only the OpenCV resource containers have

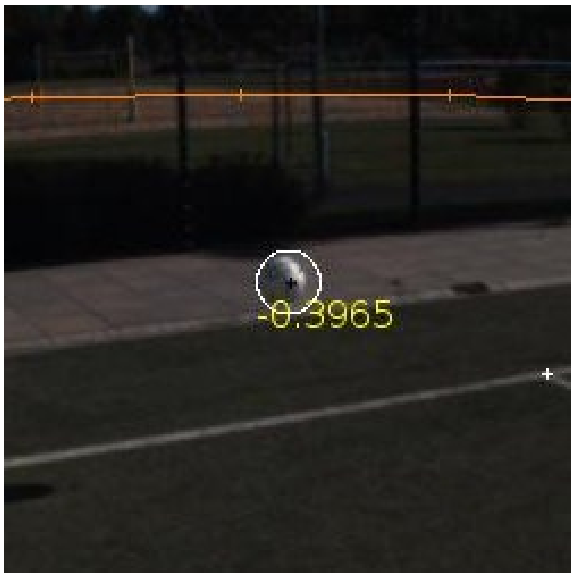


Figure 21: Incorrect start ball

been used, the internal Hough algorithm for example turned out to be not very efficient. Still, the OpenCV library includes methods for displaying these images, so it would have been better to write the complete GUI in OpenCV or maybe even to have no GUI at all.

3.2.6 Comparison with other Approaches or Applications

Even though the results of the circle detection algorithm were quite satisfying, the OpenCV circle Hough transformation was also tested. The library provides a 320x240 test image (shown in figure 22, left) for demonstrating the internal circle Hough algorithm. The circle inside this test images is detected within only a few milliseconds of time (shown in figure 22, right).

But if the algorithm is used with the actual scene images the results are clearly unacceptable. As shown in figure 23, neither the genuine black and white ball nor the coloured ball were recognised correctly. The documentation on this algorithm is unfortunately short, there are only two parameters that influence the algorithm. A small test application was developed in order to examine the algorithms behaviour, but without having a documented method for determining these "magic" parameters no satisfying results could be achieved.

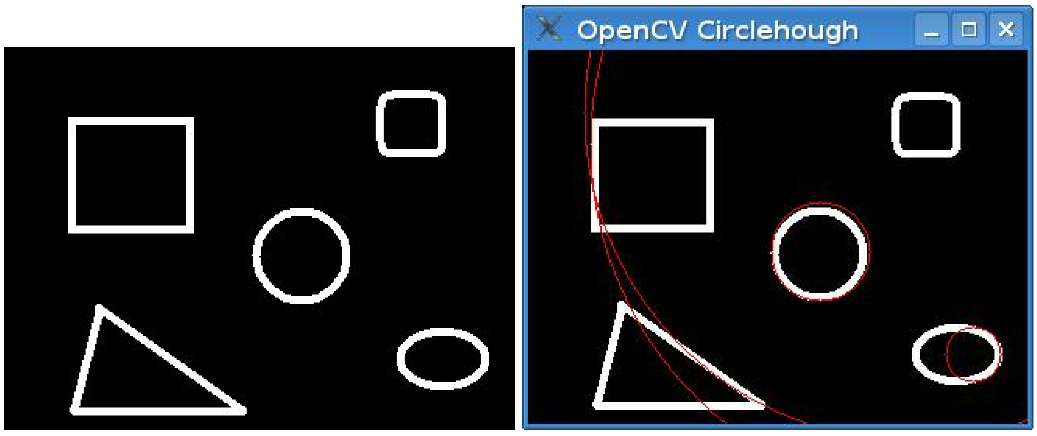


Figure 22: OpenCV circle Hough transformation example

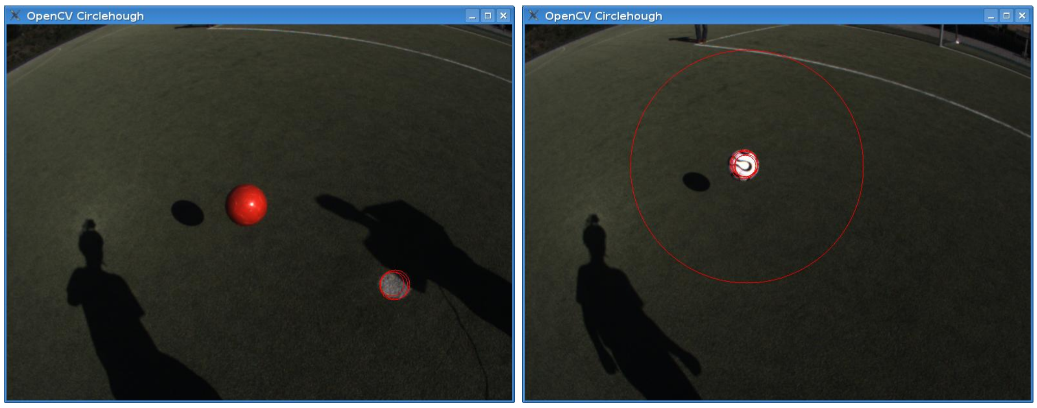


Figure 23: OpenCV circle Hough transformation results

4 Results

4.1 Evaluation

Since it is difficult to display test images in sufficient resolution within this document, the entire test results have been added to the attached DVD as full quality images. Also note that the following images have been slightly brightened and that their contrast was slightly increased. This was only done for technical reasons in order to have a well readable printed version.

4.1.1 Test Scene 1: Close Pass with a red Ball

Course of Events The observer starts at the penalty point facing the goal directly. Slowly turning his view left, he awaits a pass from about 10 meters away. The ball is kicked in the direction of the observer who attempts to stop the ball with his right foot while following the trajectory of the ball for the entire flight time. The observer is moving casually backwards in order to line up himself with the passing player and the ball. Table 6 illustrates the scene's course of action while table 7 summarises the overall results for this test.

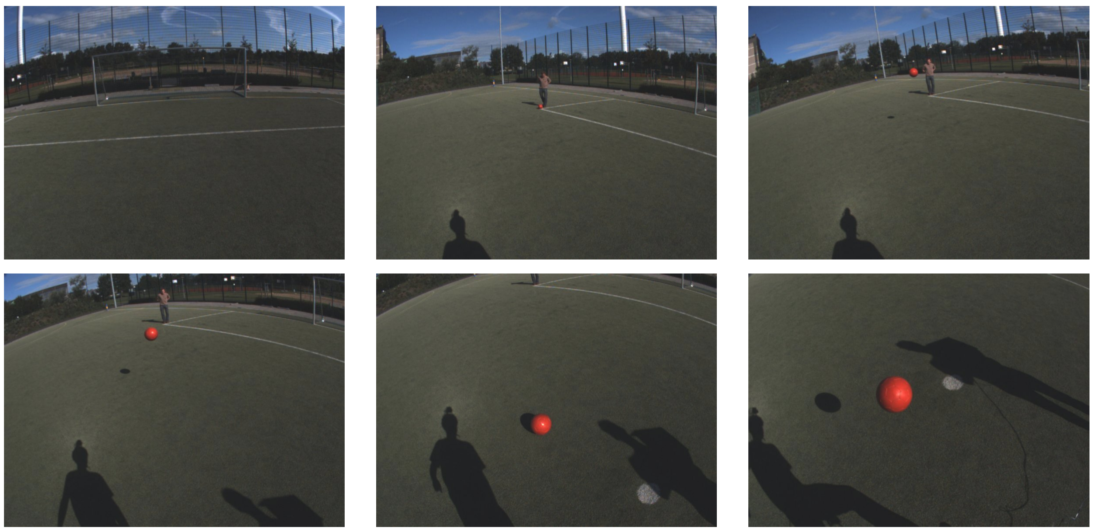


Table 6: Scene 1: A close pass with a red ball

Duration	2.94 seconds
Frames	159
Ball	red
Start ball detection	5 frames after ball leaves ground
Number	11

Table 7: Scene 1 results

Analysis The motion detector is able to find the flying ball almost exactly at the time it enters the air. Since the ball is only about 10 meters away from the observer, it is lying below the horizon and the movement is therefore be considered as a possible ball. The optimiser using the trajectory and image processing data combination did not alter the recognised ball to a worse result, which is an indication for the correctness of the proposed system. The scaling factors for the image processing error fraction have been set to be around the same bounds as the errors from the dynamic model functions.

4.1.2 Test Scene 2: Far Pass with a red Ball

Course of Events The observer again starts at the penalty point facing the goal directly. Slowly turning his view left he awaits a pass from about 20 meters away. The ball is kicked in the direction of the observer who follows the trajectory of the ball while again casually aligning himself with the passing player and the ball.Table 8 illustrates the scene's course of action while table 9 summarises the overall results for this test.

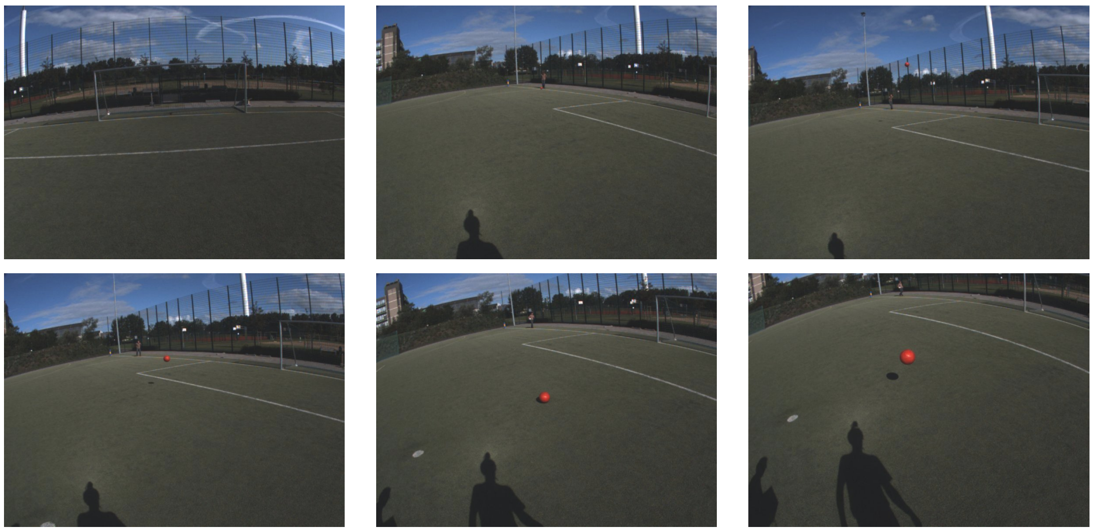


Table 8: Scene 2: A far Pass with a red Ball

Duration	4.95 seconds
Frames	269
Ball	red
Start ball detection	112 frames after ball leaves ground
Number	20

Table 9: Scene 2 results

Analysis The increased distance between ball and observer has a noticeable impact on the ball recognition in this scene. The ball is again lying below the horizon, but the motion detector is unable to detect the movement at such a great distance. At the time the detector might be able to locate the ball, it is already flying above the horizon and therefore the movement is ignored until the ball drops below the horizon. Unfortunately, this happens not until the ball has already completed about half of its trajectory.

4.1.3 Test Scene 3: Authentic Corner Kick with a black and white Ball

Course of Events The observer starts at the right corner of the goal box facing the right goal pole. Slowly turning his view left, he awaits a pass from about 35 meters away. The ball is kicked in the direction of the observer who follows the ball trajectory without moving himself. The ball passes the goal in a high flight path and hits the ground about 5 and 10 meters to the right of the observer. Table 10 illustrates the scene's course of action while table 11 summarises the overall results for this test.

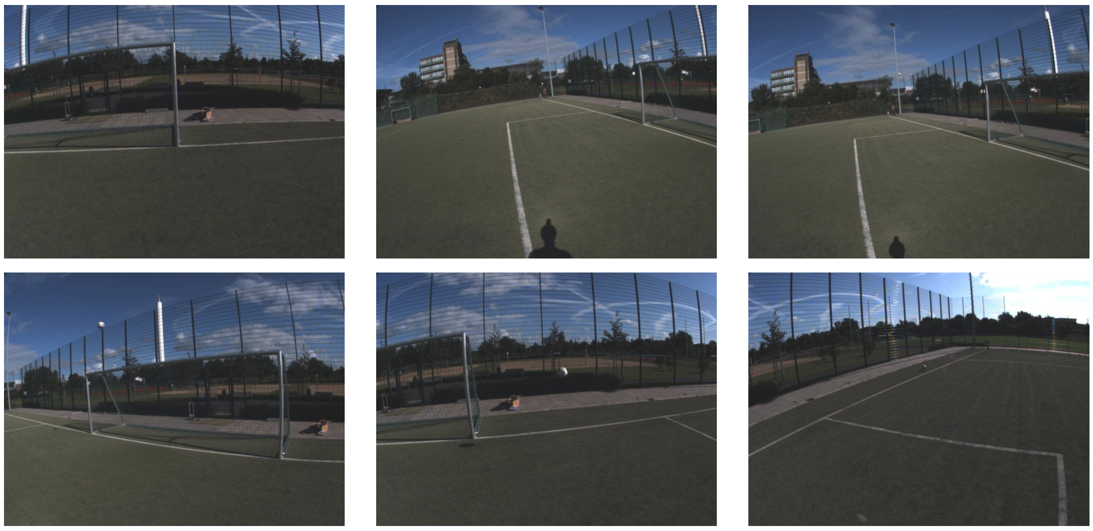


Table 10: Scene 3: An Authentic Corner Kick with a black and white Ball

Duration	5.3 seconds
Frames	287
Ball	black & white
Start ball detection	106 frames after ball leaves ground
Number	27

Table 11: Scene 3 results

Analysis The results resemble the results of test scene 2. Detection of a start ball is difficult because of the distance to the ball. At the beginning of the trajectory, the ball is still to far away to be detected and once it reaches the detection range it is already above the horizon and therefore not a valid starting ball anymore. It is immediately detected once it falls below the artificial horizon.

4.1.4 Test Scene 4: Very far Corner Kick with red Ball

Course of Events The observer again starts at the right corner of the goal box facing the right goal pole. Slowly turning his view left, he awaits a pass from about 35 meters away. This time the ball is kicked in the direction of the penalty point while the observer follows the trajectory of the ball and very slowly moves in the same direction the ball is flying. The ball hits the ground about twice in about 5 meters distance to the observer. Table 12 illustrates the scene's course of action while table 13 summarises the overall results for this test.

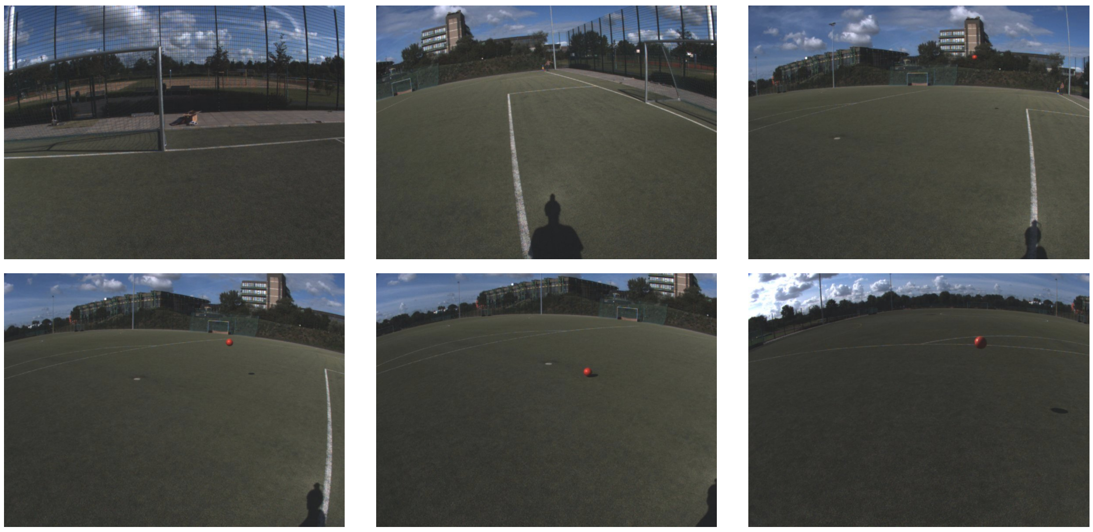


Table 12: Scene 4: Very far corner kick with red ball and long flight path

Duration	5.5 seconds
Frames	298
Ball	black & white
Start ball detection	105 frames after ball leaves ground
Number	33

Table 13: Scene 4 results

Analysis The late detection resembles that circumstances of the previously evaluated scene. Another reason why the start ball detection is bad in this scene is because of the fact that the external vision data (landmarks, etc.) is only provided from the time the ball bounces for the first time. The start ball detector was restricted to not detecting start balls

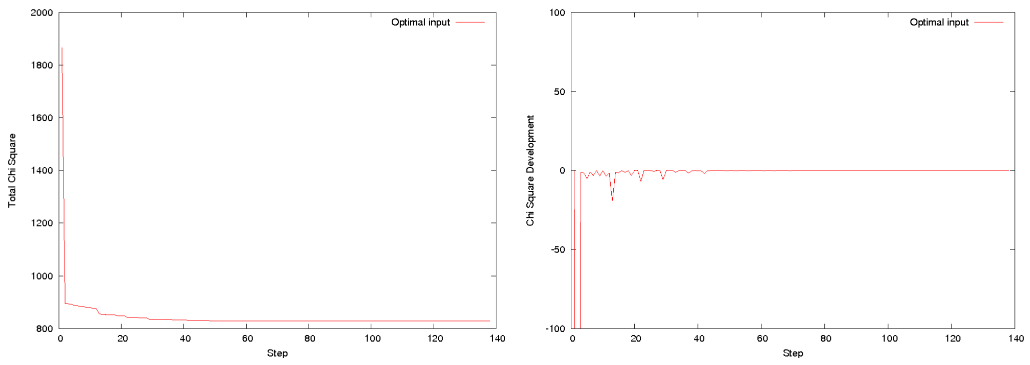


Figure 24: Total error for optimal values (left) and development per step (right)

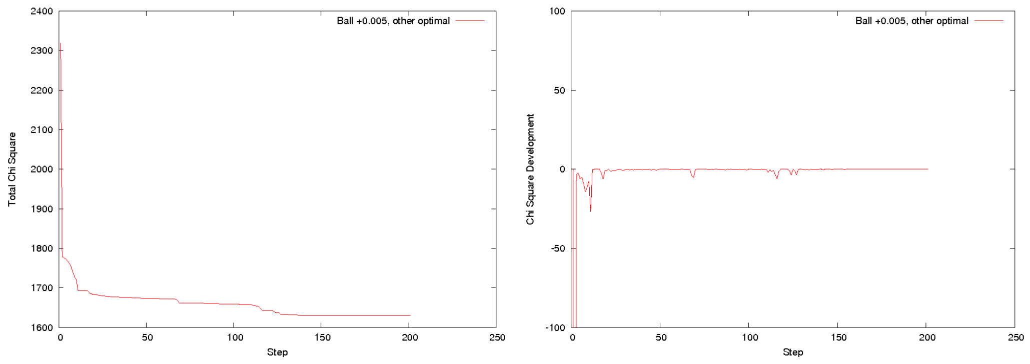


Figure 25: Total error for slightly altered values (left) and development per step (right)

for frames it does not have any external vision data for. Otherwise the optimiser would later not be able to calculate the error in frames for which it has no external data available.

4.2 Optimiser Correctness and Overall Evaluation

To prove that the optimiser actually reduces the errors in the ball flight trajectory, the error reduction of the second test scene was logged in detail. The following results resemble test runs with the other scenes.

The optimiser improves the errors in the trajectory for optimal external data (shown in figure 24) and for data that was altered to be continuously worse. The ball position was therefore changed more significantly than the remaining vector data.

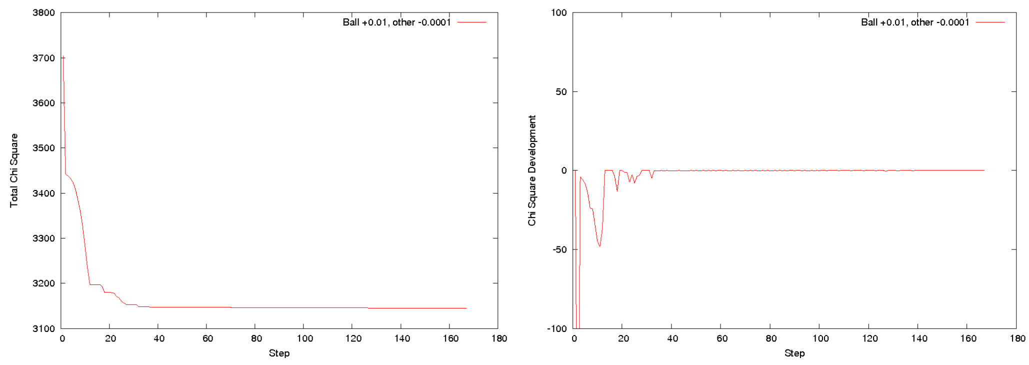


Figure 26: Total error for significantly altered values (left) and development per step (right)

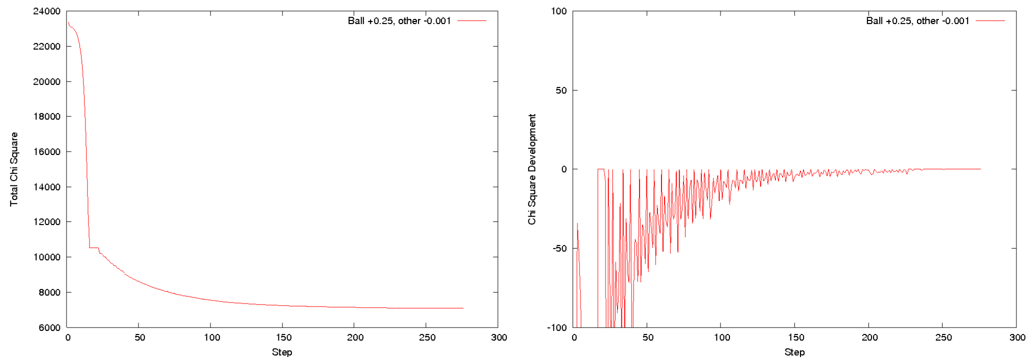


Figure 27: Total error for highly altered values (left) and development per step (right)

As the images in figure 28 and 29 show, the ball position cannot be corrected sufficiently once the input values (ball position in world coordinates) are significantly altered. Even though the error is constantly reduced, the gradient descent step width reaches the termination limit after several hundred of steps, which causes the optimiser to abort. As the images show, the position of the estimated ball is still a little bit off compared to the real ball in the image.

The reason for this effect is that the image processing related evaluators cannot output any useful values once the position of the ball is too far of the actual ball. When used on free ground, it is unlikely that the evaluator will be able to return the ball position back to the correct location because the circle values on free ground will be more or less the same.

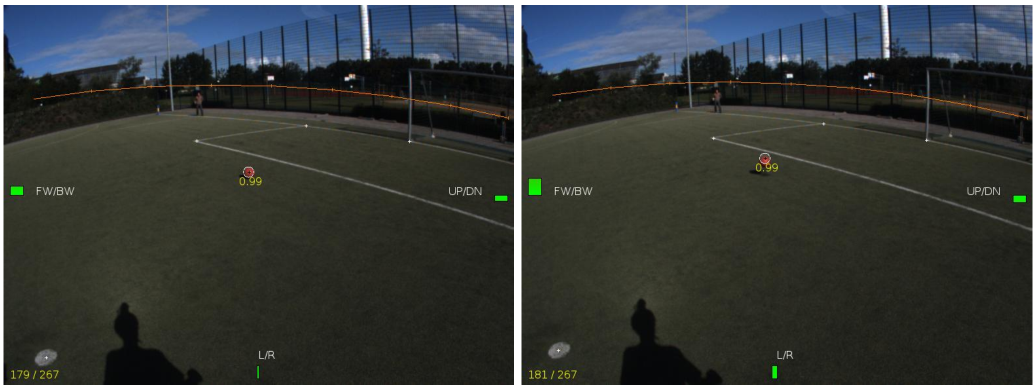


Figure 28: No modification of external input values (left), ball position changed by 5cm (right)

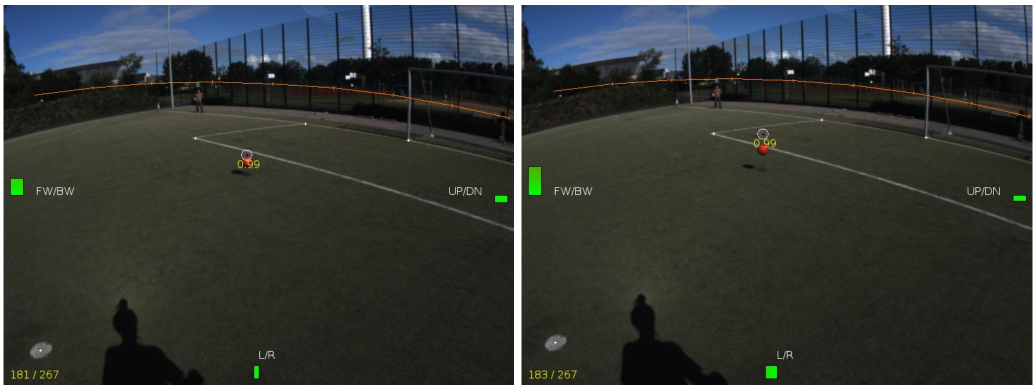


Figure 29: Ball position changed by 10cm and remaining values altered by 0.001 (left), ball position changed by 25cm and remaining values altered by 0.01 (right)

4.3 Conclusion

The set goals have been achieved, although the system's performance due to the approaches that were used is far away from being effective. The ball can be recognised in the selected test scenes even though this currently takes up an incredible amount of computation time. If used with accurate external data the system will not noticeable decrease the accuracy the trajectory data. But if the initial data is significantly altered, the optimisation process is not able to correct the ball position sufficiently.

This thesis has shown that combining visual information with additional trajectory data and optimising the overall result is an interesting approach for building a more robust system. There is also still other information that could have been merged into the computation (e.g. a game state that limits the ball position) which would further improve the system's performance but was left out due to lack of time. The downside of bringing other context into play is that errors are harder to find.

During the development of the software new problems emerged quite frequently. Most of these problems for themselves were not very difficult to solve, but they still took up time to eliminate them. This was also due to the fact that even though I tried hard, I just could not really get familiar with this area of computer science and only started understanding it when the time for development was already exhausted. New aspects that I did not think about before, called for changes at several software components which in most cases also were not too hard to introduce by themselves. Taken together, even though the software was designed to be clearly structured and easy to extend, the lack of time sometimes called for quick solutions which corrupted the original clean design. To sum up all problems during this thesis one can actually say: I underestimated the complexity of this thesis, made several wrong decisions when choosing tools and spent too much time on irrelevant tasks, for example implementing my own Kalman Filter instead of just using Kurlbaum's.

Still, the developed system proved that the original setup for dealing with the inferior mechanical capabilities of state of the art humanoid robots is useful when concentrating on the software aspects and that this approach is worth of being further investigated. Using only image processing related methods (movement between frames combined with circle detection on a limited surrounding area) on the other hand also produced good results. As final conclusion the usage of optimisation should be a part of the overall system but other methods should definitely be considered, too.

The developed system furthermore served as a visualiser for Oliver Birbach's prediction data for a short period of time. By the time this and his work was finished, this data could already be visualised by his own software, too. The developed head-camera system was also already used to gather new data for SLAM experiments and could easily be used in future lectures or other related experiments.

Future work should also consider some technical insights of this project. C++ is a very versatile and powerful programming language, but if this thesis would have to be done again, it should definitely use MATLAB [32] instead. Dealing with vectors, matrices, derivations and related mathematical operations can be done a lot easier in MATLAB than in C++. Even though most students (like myself) might not be familiar with the MATLAB software the amount of time saved by using MATLAB compared to the amount of time necessary for learning it would benefit the usage of MATLAB. Debugging the code (especially the Unscented Kalman Filter) took up a very large amount of time and would be a lot more comfortable in MATLAB. Since Oliver Birbach's related thesis was also done in MATLAB, a lot of work and time could have been saved because results could have been exchanged in an easier way.

4.4 Future Work

Hardware The camera-inertial system should be sufficient for the first future applications. Still, it would be possible to invert the synchronisation signal so that the inertial sensor may trigger the camera and not vice versa. In the current state of development the inertial sensor is triggered by the camera which results in only 54 (of a maximum of 400) sensor measurements per second. This was sufficient for both Oliver Birbach's and my thesis but might not be for future applications. It would also be interesting if this did actually cause a noticeable improvement of performance. We originally intended to have twice as many sensor measurements as pictures per second, but due to the lack of time and expertise hardware changes in the inertial sensor were not made. The manufacturer's technical support confirmed that it would be possible to change the SyncIn to a SyncOut signal with a few simple hardware modifications. This should only be done if one has good knowledge about electronics and would also void any remaining warranties for the sensor. If the sensor actually triggers the camera, the maximum FPS of the camera would be about one fifth of the maximum sensor frequency. A possible solution might be an intermediate microcontroller that reads the sensor synchronisation signals and only forwards every second signal to the camera.

Realtime The Gradient Descent optimising algorithm which is currently be used is terribly slow and definitely not the best choice for this application. It should be replaced with another kind of faster algorithm which would decrease the computation time dramatically.

For a realtime system a clean and efficient redesign and/or portation of the available code would of course also be possible. The use of parallel hardware or GPU implementation might also be of interest.

Scenes Except for the throw-in scenes, almost every recorded scene has the ball lying on the ground at the start and then flying in some sort of curve. Once this can be recognised satisfactorily other scenes with the ball already in the air or simply rolling on the field could be interesting. For the actual long-term RoboCup objective this would be definitely be necessary.

5 Acknowledgements

I would like to thank my tutor Dr. Udo Frese for his extensive help and support during my thesis. The used core algorithm for rating circles was part of his experiments, the ported Unscented Kalman Filter was part of one of his courses that I participated in and the Gradient Descent algorithm was also integrated according to one of his experiments.

Also, I would like to thank my fellow students and friends Armin Burchardt as well as Sebastian Schleusener for their help during the data acquisition experiments. Special thanks go to Oliver Birbach for the excellent collaboration with our shared material and his very helpful support during development and tests of my software. The developed system uses the dynamic model and Chi square (dynamic model scene error) computation algorithm from his thesis.

I am also thankful for the help of Jörg Kurlbaum who provided a good basis for my thesis (that I unfortunately did not use like I should have) and also helped me getting started with all of this. The software that was used to record the scene data was part of this thesis and was only slightly modified by Oliver Birbach and myself.

Furthermore Dr. Christoph Zetsche and Dr. Kerstin Schill also provided some useful thoughts and ideas that contributed to my thesis. Last but not least I would like to thank my parents who always supported me during my studies and my Iaido Sensei René Neufang and Senpai Jörg Gissel who kept me motivated during my thesis and my friend Christian Lauterbach for final proofreading.

Thank you!

List of Figures

List of Tables

References

[1] RoboCup Federation. Official RoboCup Homepage, 2007. http://www.robocup.org (last visited 28.12.2007).

[2] Honda Worldwide. The Honda Humanoid Robot ASIMO, 2008. http://world.honda.com/ASIMO/ (last visited 25.12.2007).

[3] U. Frese, B. Bäuml, S. Haidacher, G. Schreiber, I. Schaefer, M. Hähnle, and G. Hirzinger. Off-the-Shelf Vision for a Robotic Ball Catcher. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1623–1629, 2001. http://www.informatik.uni-bremen.de/~ufrese/published/freseiros.pdf.

[4] Hawk-Eye Innovations. Official Hawk-Eye Homepage, 2007. http://www.hawkeyeinnovations.co.uk (last visited 28.12.2007).

[5] J. Kurlbaum. Verfolgung von Ballflugbahnen mit einem frei beweglichen Kamera-Inertialsensor. Master's thesis, Universität Bremen, 2007. http://www.informatik.uni-bremen.de/~ufrese/teaching/thesis/kurlbaum_thesis_07.pdf (last visited 01.07.2007).

[6] R. E. Kalman. A New Approach to Linear Filtering and Prediction Problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[7] E. Kraft. A Quaternion-based Unscented Kalman Filter for Orientation Tracking. *Proceedings of the Sixth International Conference of Information Fusion*, 1:47–54, 2003.

[8] A. Hoppe, J. Kurlbaum, B. Mohrmann, M. Poloczek, D. Reinecke, T. Röfer, and U. Visser. Bremen Small Multi-Agent Robot Team (B-Smart) Team Description for RoboCup 2003. *7th International Workshop on RoboCup 2003 (Robot World Cup Soccer Games and Conferences)*, 2004. To appear.

[9] J. Bruce, S. Zickler, M. Licitra, and M. Veloso. CMDragons 2007 Team Description, School of Computer Science Carnegie Mellon University, 2007. http://reports-archive.adm.cs.cmu.edu/anon/home/ftp/2007/CMU-CS-07-173.pdf (last visited 03.01.2008).

[10] T. Röfer, J. Brose, E. Carls, J. Carstens, D. Göhring, M. Jüngel, T. Laue, T. Oberlies, S. Oesau, M. Risler, M. Spranger, C. Werner, and J. Zimmer. GermanTeam 2006. *RoboCup 2006: Robot Soccer World Cup X Preproceedings*, 2006.

[11] G. Fritz, L. Paletta, and H. Bischof. Object Recognition Using Local Information Content. *Proceedings of the 17th International Conference on Pattern Recognition*, 2:15–18, 2004.

[12] G. Fritz, C. Seifert, L. Paletta, and H. Bischof. Rapid Object Recognition from Discriminative Regions of Interest. http://citeseer.ist.psu.edu/725199.html (last visited 14.10.2007).

[13] C. Hudelot and M. Thonnat. A cognitive vision platform for automatic recognition of natural complex objects. *15th IEEE International Conference on Tools with Artificial Intelligence*, pages 398–405, 2003. http://citeseer.ist.psu.edu/717874.html (last visited 14.10.2007).

[14] V. Vu, F. Bremond, and M. Thonnat. Video interpretation: human behaviour representation and on-line recognition. *"The Sixth International Conference on Knowledge-Based Intelligent Information and Engineering Systems"*, 2002. http://citeseer.ist.psu.edu/vu02video.html (last visited 16.02.2008).

[15] O. Birbach. Accuracy analysis of camera-inertial sensor-based ball trajectory prediction. Master's thesis, Universität Bremen, 2008. http://www.informatik.uni-bremen.de/~ufrese/teaching/thesis/birbach_thesis_08.pdf (pending).

[16] T. Laue, T. Schindler, F. Penquitt, A. Burchardt, O. Birbach, C. Elfers, and K. Stoye. B-Smart (Bremen Small Multi-Agent Robot Team) Team Description for RoboCup 2006, 2006. http://www.b-smart.de/wiki/images/8/8d/B-Smart_TDP2006.pdf (last visited 01.01.2008).

[17] Sir W. Hamilton. Lecture on Quaternions, 1853. http://historical.library.cornell.edu/cgi-bin/cul.math/docviewer?did=05230001&seq=9 (last visited 05.01.2008).

[18] J. H. Conway and D. A. Smith. *On Quaternions and Octonions: Their Geometry, Arithmetic, and Symmetry.* B&T Publishing, 2003.

[19] Wikipedia. Quaternion, 2008. http://en.wikipedia.org/wiki/Quaternion (last visited 05.01.2008).

[20] R. Hanek and M. Beetz. The Contracting Curve Density Algorithm: Fitting parametric curve models to images using local self-adapting separation criteria. *International Journal of Computer Vision*, 59:233–258, 2004.

[21] G. M. Schuster and A. K. Katsaggelos. Robust line detection using a weighted MSE estimator. *Proceedings of the 2003 IEEE International Conference on Image Processing (ICIP), Barcelona, Spain*, 2003.

[22] A. Ben-Israel and T. N. E. Greville. *Generalized Inverses: Theory and Applications*. Springer, New York, 2003.

[23] P. V. C. Hough. Method and Means for Recognizing Complex Patterns, 1962. U.S. Patent 3069654.

[24] D.H. Ballard. Generalizing the Hough Transform to Detect Arbitrary Shapes. *Pattern Recognition*, Vol. 13(No. 2):111 – 122, 1981.

[25] H. K. Yuen, J. Princen, J. Illingworth, and J. Kittler. Comparative study of hough transform methods for circle finding. *Image Vision Computing*, 8(1):71–77, 1990.

[26] H. Li, M. A. Lavin, and R. J. Le Master. Fast Hough transform: A hierarchical approach. *Computer Vision, Graphics, and Image Processing*, Volume 36(Issue 2-3):139 – 161, 1986.

[27] Intel. Open Computer Vision Library, 2008. http://www.intel.com/technology/computing/opencv/index.htm (last visited 07.02.2008).

[28] U. Frese. Camera Sensor Model, 2007. Not published.

[29] J. A. Snyman. *Practical Mathematical Optimization: An Introduction to Basic Optimization Theory and Classical and New Gradient-Based Algorithms*. Springer, 2005.

[30] Trolltech. Qt: Cross-platform rich client development framework, 2008. http://trolltech.com/products/qt.

[31] The MPlayer Project. MPlayer/MEncoder Homepage, 2008. http://www.mplayerhq.hu/ (last visited 08.02.2008).

[32] The MathWorks Deutschland. MATLAB - The Language of Technical Computing, 2008. http://www.mathworks.de/products/matlab/ (last visited 16.02.2008).

A Software

The enclosed CD contains:

1. A digital version of the final thesis paper with all figures in original resolution
2. The repository of the actual proposed system with all its sources and in binary form together with the user and developed documentation (*/thesis/repository/code*).
3. Some of the cited and several other interesting papers on related topics as PDF versions (*/thesis/papers*).
4. The four test scenes which were analysed within the *Test* section and some (intermediate) results in image or video form (*/thesis/scenes/, thesis/scenes/results*).
5. Miscellaneous material like hardware documentation, the original MATLAB sources of the used dynamic model functions and the Libertine fonts which were used for this document (*/thesis/misc/*).