**UNIVERSITY OF OSLO**
**Department of Informatics**

# Bagadus: next generation sport analysis and multimedia platform using camera array and sensor networks

Masteroppgave

Simen Sægrov

# Bagadus: next generation sport analysis and multimedia platform using camera array and sensor networks

Simen Sægrov

**Abstract**

Today, a large number of (elite) sports clubs spend a large amount of resources to analyze their game performance, either manually or using one of the many existing analytics tools. In the area of soccer, there exist several systems where trainers and coaches can analyze the game play in order to improve the performance. However, most of these systems are cumbersome and relies on manual work from many people and/or heavy video processing.

In this thesis, we present Bagadus, a prototype of a soccer analysis application which integrates a sensor system, soccer analytics annotations and video processing of a video camera array. The prototype is currently installed at Alfheim Stadium in Norway, and we demonstrate how the system can follow and zoom in on particular player(s), and search for and playout events from the games using the stitched panorama video and/or the camera switching mode.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Background and motivation

Today, a large number of (elite) sports clubs spend a large amount of resources to analyze their game performance, either manually or by using one of the many existing analytics tools. In the area of soccer alone, there exist several systems where trainers and coaches can analyze the game play in order to improve the performance of the players and the team. For instance, Interplay-sports [1] has been used since 1994, where video-streams are manually analyzed and annotated using a soccer ontology classification scheme. In this system, trained and soccer-skilled operators tag who has the ball, who made a pass, etc.

Another commonly used system is ProZone [2] – a system that automates some of this manual notation process by video-analysis software. In particular, it quantifies movement patterns and characteristics like speed, velocity and position of the athletes. In this respect, Valter et. al. [3] conducted an empirical evaluation of deployed ProZone systems at Old Trafford in Manchester and Reebook Stadium in Bolton, and concluded that the video camera deployment gives an accurate and valid motion analysis. Similarly, STATS SportVU Tracking Technology [4] uses video cameras to collect the positioning data of the players within the playing field in real-time. This is further compiled into player statistics and performance. As an alternative to video analysis, which often is inaccurate and resource demanding, ZXY Sport Tracking (ZXY) [5] uses global positioning and radio based systems for capturing performance measurements of athletes. Using a sensor system, ZXY captures a player's orientation on the field, position, step frequency and heart rate frequency with a resolution of samples up to 20 times per second. Using these sensor data, ZXY's system can present player statistics, like speed profiles,

accumulated distances, fatigue, fitness graphs and coverage maps, in many different ways like charts, 3D graphics and animations.

To improve the game analytics, video becomes increasingly important where the real game events are replayed. However, the integration of the player statistics systems and video systems still requires a large amount of manual work, e.g., events tagged by coaches or other human expert annotators must be manually extracted from (or marked in) the videos. Furthermore, connecting the player statistics to the video also require manual work.

Search engines today have limited capabilities in regard to video search and this is mainly related to the difficulties of indexing videos. Although automatic video analysis methods exist, they have several limitations in regards to both precision and resource requirements. Video indexing often requires annotations in the video itself and while automatic annotation systems exist [6], humans are often needed to create precise annotations. In soccer, matches are often commented live on the Internet, and prototypes like DAVVI [7] can utilize this to annotate videos. However, very fine grained annotations like the location a soccer player on every frame is a task too comprehensive for a human.

Video tracking is the process of locating a moving object over time, using a live video stream from a camera or a recorded video stream. Video tracking algorithms have been researched for quite some time, and a real-time video tracking system was proposed as early as in 1980 [8]. In contrast to the tracking system of 1980s which depended on special purpose processors and architectures, the current algorithms could achieve real-time tracking on general purpose hardware. Several techniques and algorithms exist, and they all have their strengths, weaknesses and applications. While these algorithms is not in the scope of this thesis, it is important to know they are quite complex and computationally expensive [9]. This is especially true when working on high definition video and high frame rates, which is typical for today's sport events. Adding to the complexity of tracking objects, is identifying who or what we are tracking and uniquely identifying them. In certain cases this is a soluble problem, for instance one could identify a tracked car by its license plate, but it is harder to identify a tracked person. Several automatic methods exist for identifying persons in video [10] [11] but accuracy and performance is not always what is needed, and especially if there are multiple subjects in a video frame.

In soccer, video is used extensively as an entertainment medium through broadcasting companies but also as an analysis tool for managers and coaches. The most commonly used process of creating videos of interesting events that coaches use today is cumbersome and fairly old fashioned compared to what is possible with today's technology. In the Norwegian soccer club Tromsø IL (TIL), the process consist of using pen and paper to register interesting events,

followed by manually editing the broadcast that the TV companies produce. While this is a perfectly working solution, it has it drawbacks in terms of both time and effort needed. One could easily imagine a scenario where an important situation happens during the first half of a soccer game, and the coach want to present this to the team during half-time. This is not possible with the current process. Another drawback with the current solutions is the cost requirement and in smaller soccer clubs like TIL, where budgets are relatively small compared to larger European clubs, cost is always an issue. In TIL, the current solution depends on cameras used by the broadcasting companies, which is only present during official matches, and are also dependent on cameramen controlling them. When using multiple cameras, it is possible to cover the entire field without the need for a cameraman to control the cameras.

As already mentioned, video is used extensivly in soccer as an entertainment medium. In Norwegian soccer, the trends show that fans are attending the game to a lesser extent than before. In the period 2007-2010, Rosenborg (one of Norway's most popular soccer teams) lost a third of its spectators while TV2 reported an increase in viewers during the same period [12]. At the same time, more and more services delivering video are breaking with the traditional television broadcasting and instead creating a more personalized user experience where the user can decide what to watch. We will later see how we can use video tracking to create a personalized user experience and also how video tracking can be used to create search-able meta content.

## 1.2 Problem Definition

The current soccer analysis tools of today relies on manual work and/or complex and expensive computations, leaving them ill suited for real-time scenarios. In this thesis, we will present a solution for tracking objects in multiple synchronized cameras by integrating a video sub-system with a sensor network. By using sensor data for tracking, you almost completely eliminate the complex and expensive step of finding the actual object you want to track, because a players position on the field is given by the sensor system. In the context of tracking a player in multiple synchronized cameras, we want to investigate a method for following players through the field by automatically switching camera streams.

In order to provide an overview of the field and an additional tool for the coaches, we want to stitch together frames from multiple synchronized cameras in order to create a high resolution panoramic view, and we will investigate how this can be used in combination with player tracking and digital zooming in order to create video sequences that follows players closely. In the context of video stitching, we will drop our real-time goal and rather look at methods and

problems of video stitching.

We will investigate some of the capabilities that real-time video tracking and sensor data provide in regards to creating searchable video content and how this can help us create personalized user experiences. At last, we will look at some of the applications real time video tracking gives us, and we will see how this can be applied in a publication/subscription scenario and in a home entertainment setting. Also, we will see how this can be used as a tool for expert user groups like coaches and managers.

In this thesis, we present a system called Bagadus which *integrates* a camera array video capture system with the ZXY Sports Tracking system for player statistics and a system for human expert annotations. Bagadus allows the game analytics to automatically playout a tagged game event or extract a video of events extracted from the statistical player data like all sprints at a given speed. Using the exact player position, we can also follow individuals or groups of players. The videos are presented using a stitched panorama video or by switching cameras. Our prototype is applied at Alfheim Stadium (Tromsø IL, Norway), and we will here use a dataset captured from a Norwegian premier league game to demonstrate our system.

## 1.3 Limitations

When working on large data sets like in this thesis there are many hardware requirements and limitations in regard to bandwidth. Each of the cameras we use produce roughly 70 MB/sec of data, and this creates bottlenecks both on network transfers and hard drives. We will identify these bottlenecks, and in some cases propose solutions and workarounds, but this is not the primary focus of this thesis.

Also, the sensors deliver sensor data at a rate of 20 Hz, and the 22+ sensors pushes the database that ZXY uses to its limit. In this thesis, we have worked on a offline database and the sensor data is not provided in real-time by ZXY. Thus, we are leaving database performance issues out of scope. Another goal has been to build a system that works in real-time, with the exception of video stitching and having the entire pipeline of our system is also out of scope.

## 1.4 Research Method

In this thesis, we will design, implement and evaluate a working prototype which demonstrates some of the capabilities such a system gives. Our system will be deployed and tested in a real-

life scenario, in which the actual users of such a system can interact. Our approach is based on the *Design* methodology as specified by the ACM Task Force on the Core of Computer Science [13].

## 1.5 Main Contributions

On Alfheim stadium, we have installed cameras and developed video recording software that can record synchronized videos from multiple cameras. In addition, we have made an application that can play theses synchronized recorded videos, where the videos are rectified and debarreled in real-time – enabling us to track players in the videos. Player tracking is made possible, by using our ZXY database interface for delivering coordinates for players. We also introduced a video stitcher which is integrated into the application. With our demo application, we have showed that it is possible to create an almost fully automatic soccer analysis system, without the need of manual post processing.

Doing research that commercial partners can benefit from, with real life deployment and testing has been a huge motivation throughout our work.

A video demo describing Bagadus has been made and can be viewed at [1].

During the thesis, we have had close contact with representatives of both Tromsø IL and ZXY Sports Tracking. Through our prototypes and demos, we have showed them some of the potential that this technology gives and ZXY Sports Tracking has already showed interest in commercializing this and integrating it into their products.

## 1.6 Outline

Bagadus consist mainly of three components, *video capture*, *video tracking* and *video stitching* and this thesis is structured likewise. In chapter 2, we will look at how we can capture and synchronize multiple camera streams. Chapter 3 introduces tracking of players in multiple synchronized cameras, and in chapter 4, we will look into different alternative of how we can stitch multiple cameras into one panoramic view. In chapter 5 we will see how we can integrate these components into a tool for soccer analysis and as a tool for personalized media entertainment. Each chapter contains different implementation approaches and its own discussion and

[1] http://home.ifi.uio.no/paalh/videos/bagadus.mov

evaluation. Finally, in chapter 6, we summarize and conclude the thesis.

# Chapter 2

# Video capture

In this chapter, we will look at the first step in our pipeline, namely the video capture part. One of our areas of focus was to make a system that did not rely on manual control and manual post processing. In contrast to the Interplay Sports system where equipment has to be set up prior to the game, live annotated during the game and then further post processed after the game, we wanted to make a system where you could basically press record just before the game started, and then stop when you wanted to without any other intervention. Also, video should be available on the fly while recording via a web server. In addition, we wanted (or we were forced) to use fairly cheap equipment, which again lead to some interesting problem areas which we will highlight in this chapter.

## 2.1 Hardware and setup

As previously mentioned we wanted to use fairly cheap common hardware and no expensive special purpose hardware. The primary reason for this was what it always comes down to – money.

In our setup at Alfheim stadium, we have four "Basler acA1300 - 30gc" [14] industrial Ethernet cameras mounted with Kowa 3.5mm wide angel lenses. Each Camera is connected and controlled by its separate computer, however, it is possible to control multiple cameras with a single computer given that you have enough ethernet interfaces. The Basler camera can be seen in figure 2.1 on the following page. The Kowa wide angel lenses enables us to cover the entire soccer field and figure 2.2 on the next page, illustrates the position and the field of view for each camera. The wide angel lenses are need to cover the entire field and wide angel lenses are know

to have severe geometrical distortion. Later in this chapter, we see how and why we need to correct this.

In this thesis, we have enumerated each camera from one through four, going from left to right. Each camera is connected to a separate computer, which runs the capturing software. Ideally, the cameras would be evenly space out along the sidelines, however, the control room where the machines are, is close to center of the field and factors such as cable length on our trigger boxes (explained later) forces us to use the setup as illustrated in figure 2.2.


Figure 2.1: The Basler ACE camera [14].

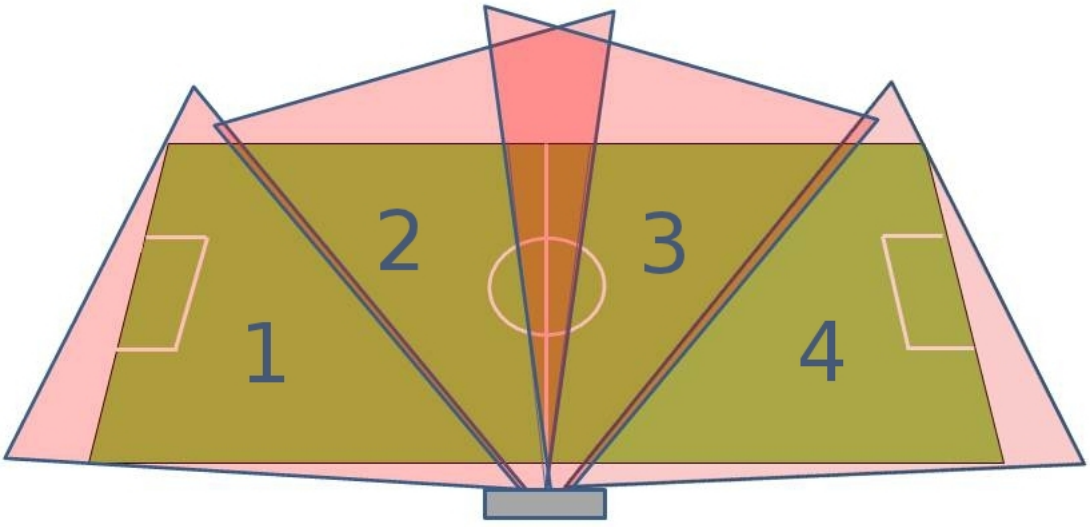


Figure 2.2: Our camera setup at Alfheim stadium.

The computers used to control the cameras and run the performance tests in this thesis has the following specifications. Intel Core i7-2600 @ 3.4 GHz, 8 GB memory and nVidia GTX 460 graphic card.

## 2.2 Northlight

The industrial cameras does not come with a software package that allows us to record video, and we needed to make our own recording software. *Northlight* is a library developed and maintained by Alexander Eichhorn and students at Simula Research Laboratory. It was started as a part of the Verdione project which is a computer vision project and we have participated in the development of this library.

The Verdione Project is a large project and its main focus is real-time video processing. Northlight aims to create a common interface between many popular open source libraries related to video and computer vision. All of the components in this thesis are built as a part of Northlight, utilizing much of the functionality already built into this project. In particular, we have used Northlight for controlling the cameras using the SDK provided by Basler [14], video encoding using x264 [15], colorspace conversion using FFmpeg [16], JPEG-compression using OpenJPEG [17] and computer vision algorithms using OpenCV [18]. In this chapter, we will given a short introduction into all these fields and how they used.

### 2.2.1 Camera control

The Basler cameras comes with a SDK for controlling the cameras. This includes, auto exposure, white balance, shutters and all the little bits and pieces that is needed in order to capture meaningful video. Auto exposure is of particular importance in our prototype, because of the varying lighting conditions in Tromsø, ranging from long summer days to dark winter evenings. While the auto exposure controls the shutter speed, a synchronization device is need in order to synchronize the shutters in the cameras. Later in this, we will look at how we can stitch together frames from multiples cameras, which calls for the synchronization of the shutters. At Simula, a electronic trigger box which sends synchronized signals to the cameras, signaling when to trigger the shutters.

Each of our cameras records video at 30 frames per second and each video frame in our current prototype is a 8 bit color image with a resolution if $1280 \times 960$ pixels, consisting of three channels. The cameras captures frames in a colorspace known as Y'CbCr (see section 2.2.2 on the following page). The cameras captures frames using a pixel format known as YUV422. Each frame is roughly 2.4 MB, which means that each camera produce roughly 74 MB/sec of data and having four cameras in our setups means that we generate 296 MB/sec of data and this quickly puts restrains on many areas of a computer. For instance if you want to store 90 minutes

of raw video, we need 1.6 TB which means that we either need to buy a lot of hard drives or use some form of compression. In the following section, we will introduce video, colorspaces, image- and video-compression techniques.

### 2.2.2 Color Spaces and image formats

#### The Red Green Blue Color space

Most video devices today, both input devices such as cameras and display equipment such as HDTV's, uses the Red Green Blue (RGB) color space. RGB, named after its three color channels; red green and blue, convey the information of each pixel by a triplet giving the amount of each color. For instance, using 8bit per channel, the colors red will be (255,0,0), green (0,255,0), blue (0,0,255), and cyan (0,255, 255). RGB is well suited for both capturing and displaying video. For instance, the pixel values can be mapped to the lighting sources in displays, such as phosphor dots in CRT monitors or sub-pixels in LCD panels. However, the three RGB channels carry more information than the human vision can absorb.

#### The human vision

The human vision system is actually two distinct systems, from the cells in the retina to the processing layers in the primary visual cortex. The first one is found in all mammals. The second is a complimentary system we share with other primates. The mammal system is responsible for our ability to register motion, depth and position, as well as our overall field of vision. It can distinguish acute variation of brightness, but it does not detect color. The primate system is responsible for detecting objects, such as facial recognition, and is able to detect color. However, it has a lower sensitivity to luminance and is less acute [19]. As a result, our ability to detect color is at a lower spatial resolution compared to our detection of brightness and contrast. Knowing this, we can reduce the amount of color information in the video accordingly. We loose information, but because of the limits of the human vision, the subjective video quality experienced will be the same.

#### Y'CbCr

To take advantage of the human vision in terms of video coding, we need a way to reduce the resolution of the color information while keeping the brightness and contrast intact. As we

noted above, this is not possible with RGB, where the pixel values are given solely by their color. However, we might use the derivative Y'CbCr colorspace. It works in an additional fashion similar to RGB, and transforming from the one to the other involves few computations. Instead of identifying a pixel value by its composition of amounts red, green and blue, it is identified by its brightness and color difference. Color difference is the difference between brightness (Luma[1]) and the RGB colors. Only the Chroma blue (Cb) and Chroma red (Cr) is transmitted, as Chroma green $(Cg) = 1 - (Cb + Cr)$. With brightness separated from color, we can treat them separately, and provide them in different resolutions to save space. Figure 2.3 illustrates the components of a Y'CbCr color image.

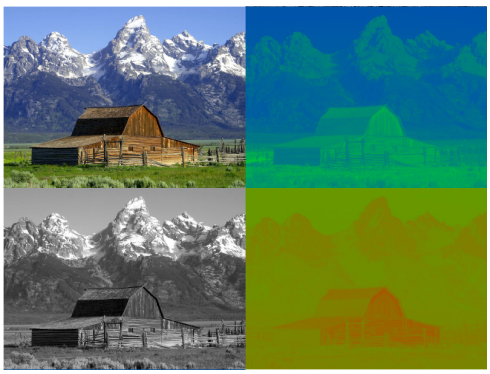

Figure 2.3: Illustration of the components of a Y'CbCr color image.

### Chroma sub-sampling

Using a lower resolution for the chroma components is called chroma sub-sampling. The default form of sub-sampling in the H.264 standard is $4:2:0$. The first number, 4, is reminiscent to the legacy NTSC and PAL standards, and represents the Luma sample rate. The second number, 2, indicates that Cb and Cr will be sampled at half the horizontal sample rate of Luma. Originally, the second and third digits denoted the horizontal subsample rate of Cb and Cr respectively, as the notation predates vertical sub-sampling. Today however, a third digit of zero now indicates half ivertical sample rate for both Cb and Cr. (For a more thorough explanation, the reader is referred to [20]). Figure 2.4 on the following page illustrates chroma sub-sampling and the relation between resolution in the different channels. Using $4:2:0$, we only use half of the luma sample size to store both chroma components. As shown in figure 2.4 on the next page, the chroma samples are only stored for every fourth luma sample.

[1]Note that the term Luma must not be confused with Luminance, as the nonlinear Luma is only an approximation of the linear Luminance [20].

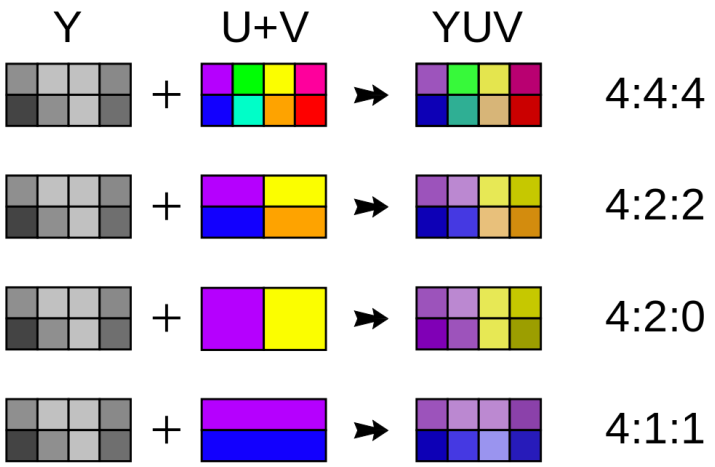


Figure 2.4: Illustration of different chroma sub samples

**Pixel arrangement**

YUV is a color model that imitates the human vision. Historically was developed to provide compatibility between color and black and white television systems. While the term YUV is not defined precisely in the technical and scientific literature, it is generally considered to include a designate family of so called luminance / chrominance colorspaces. The best way to avoid ambiguity associated with the term YUV is to refer to the concrete variant of YUV color space well defined in the internationally recognized standard documents. In this thesis when referring to YUV, we are referring to Y'CbCr as defined in the ITU-R BT.601-5 and ITU-R BT.709-5 standards of ITU (International Telecommunication Union).

YUV formats falls into two distinct groups, the packed format and the planar format. The differences lies only in the arrangement of the color channels where the packed format, Y, U, and V samples are packed together into groups and stored as single array (see figure 2.5 on the facing page). While in the planar format, each component is stored in a separate array (see figure 2.6 on the next page).

Our Basler camera captures YUV frames with $4:2:2$ sub-sampling using a packed format as shown in figure 2.5 on the facing page. The maximum resolution of our cameras is $1280\times960$ and we have included a space requirement table comparing sub-sampling (see figure 2.7 on page 14).

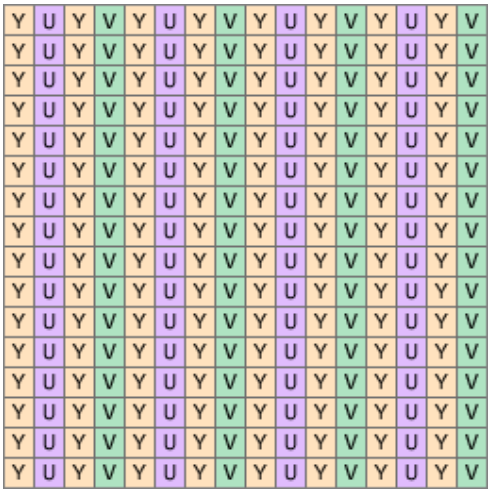


Figure 2.5: Illustration of the memory alignment in a packed YUV image.

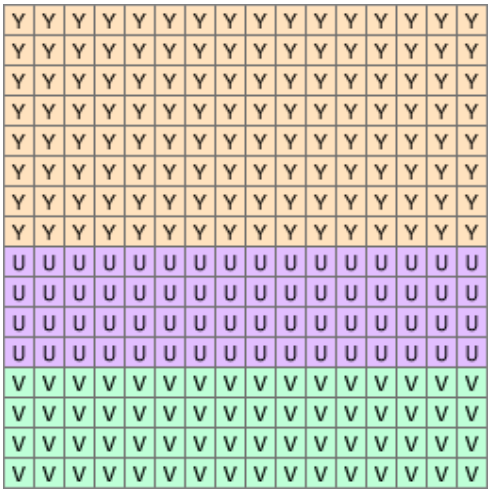


Figure 2.6: Illustration of the memory alignment in a planar YUV image.

### 2.2.3 Image compression

As we showed in the previous section, a single video frame takes $1.8MB$ even with $4:2:0$ chroma sub-sampling. Since our cameras have a frame rate of 30 fps, each camera produce roughly $55.3MB/sec$, which calls for some kind of compression before storing them to disk. Both image compression and video encoding is a vast field and we will not be diving into all the details of them, but rather give a brief introduction.

Image compression deals with reducing the amount of data required to represent a digital image by removing redundant data, i.e. reducing the number of bits needed to represent a given image. Compression methods can be lossy, when a tolerable degree of deterioration in the visual quality of the resulting image is acceptable, or lossless, when the image is encoded in its full

$$
\begin{array}{lccccccc}
Colorspace & Y & & U & & V & & \\
YUV444 & 1280*960 & + & 1280*960 & + & 1280*960 & = & 3686.4kB \\
YUV422 & 1280*960 & + & \frac{1280*960}{2} & + & \frac{1280*960}{2} & = & 2457.6kB \\
YUV420 & 1280*960 & + & \frac{1280*960}{4} & + & \frac{1280*960}{4} & = & 1843.2kB
\end{array}
$$

Figure 2.7: Comparison of YUV space requirements when reducing resolution of chrominance components

quality. The overall results of the compression process, both in terms of storage savings – usually expressed numerically in terms of compression ratio (CR) or bits per pixel (bpp) – as well as resulting quality loss (for the case of lossy techniques) may vary depending on the technique, format, options (such as the quality setting for JPEG), and the image contents.

Image compression is usually a three step process, where each step is voluntary. The steps consist of transform, quantization and coding.

**Transform**

Transforms the input data into a format designed to reduce interpixel redundancies in the input image. This operation is generally reversible and may or may not directly reduce the amount of data required to represent the image. An example of this is the discrete cosine transform (DCT) which is used in the Joint Photographic Experts Group (JPEG) format.

**Quantization**

Reduces the accuracy of the transformation's output in accordance with some pre-established fidelity criterion. Reduces the psychovisual redundancies of the input image. This operation is not reversible and must be omitted if lossless compression is desired.

**Encoding**

Creates a fixed- or variable-length code to represent the quantizer's output and maps the output in accordance with the code. In most cases, a variable-length code is used. This operation is reversible.

In this three stage process, there are four types of redundancies which are exploited in order to compress the image.

**Psychovisual redundancy**

Information in the image which we (the human eye) cannot see. The human eye is for example less susceptive to differences in color changes than intensity changes and this can be exploited by reducing the resolution in the color channels (as explained in section

2.2.2).

**Interpixel redundancy**

This type of redundancy – sometimes called spatial redundancy, interframe redundancy, or geometric redundancy – exploits the fact that an image very often contains strongly correlated pixels, in other words, large regions whose pixel values are the same or almost the same. This redundancy can be explored in several ways, one of which is by predicting a pixel value based on the values of its neighboring pixels. In order to do so, the original 2-D array of pixels is usually mapped into a different format, e.g., an array of differences between adjacent pixels. If the original image pixels can be reconstructed from the transformed data set the mapping is said to be reversible.

**Coding redundancy**

Uses statistical properties of an image by using less bits to store frequent values and more bits to store more frequent values.

In addition to these types of redundancies in images, there another one for video which most – if not all modern video encoders use. *Inter picture redundancy* - redundancy between images in time e.g redundancy between one frame and the next frame in a video stream.

### 2.2.4 Video encoding

As with image compression, video encoding is not the primary focus of this thesis, however a brief introduction is needed in order to identify some of the problems that arises when trying to play multiple streams. H.264/MPEG-4 Part 10 [21] or AVC (Advanced Video Coding) is a standard for video compression, and is currently one of the most commonly used formats for the recording, compression, and distribution of high definition video. The final drafting work on the first version of the standard was completed in May 2003. The standard is quite comprehensive and instead of giving an introduction to it, we will rather look at video encoding in general. For reference, we have included figure 2.8 on the following page, which shows a top level block diagram of the steps in an H.264 video encoder. With this in mind, we will introduce the different types of encoded frames that most modern video encoder have.

**Intra coded frame**

An intra coded frame (also know as I-frame or key-frame) is a separate encoded frame. It is not dependent on any other frame other than itself. This much like a single image encoded by itself, where standard image compression techniques are used.

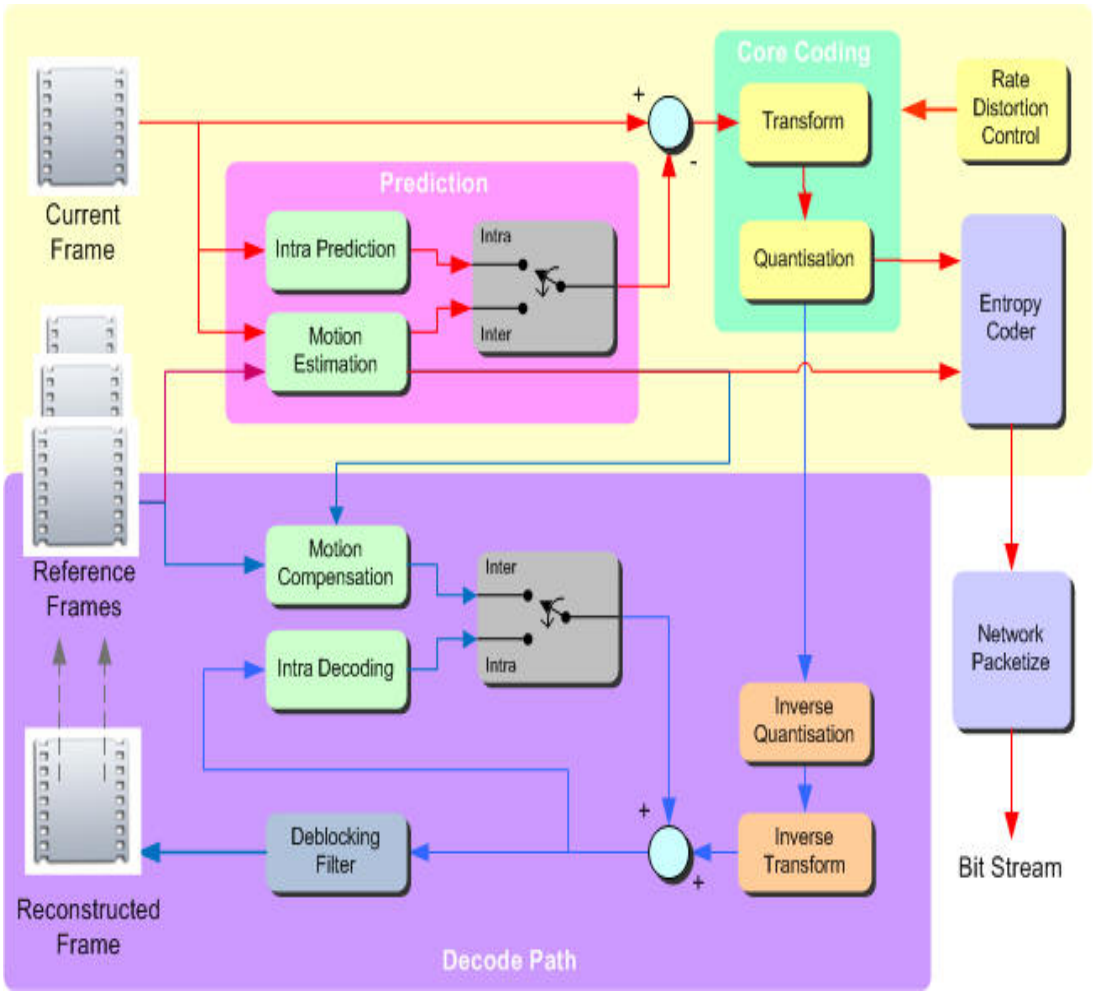


Figure 2.8: Top level block diagram of an H.264 video encoder

**Predicted frame**

A predicted frame or P-frame is dependent on a previous encoded frame to be decoded. A P-frame does only contain differences between previous and current frame and thus cannot be decoded in, and by itself.

**Bi-directional predicted frame**

A B-frame is a frame that is dependent on both preceding frames that follow. See figure 2.9 on the next page.

In terms of encoding, the frames types used does not really matter that much as long as the computer can handle the complexity and encode each frame within its time limit. I-Frames take more space, but are faster to encode compared to P-frames and B-frames. However, this means much in terms of decoding. As we will see later, the system need to be able to switch camera at any time at any point fluently. So if you are watching video from camera and hit a button to switch to the next camera, we want to be able to immediately see the next frame from new

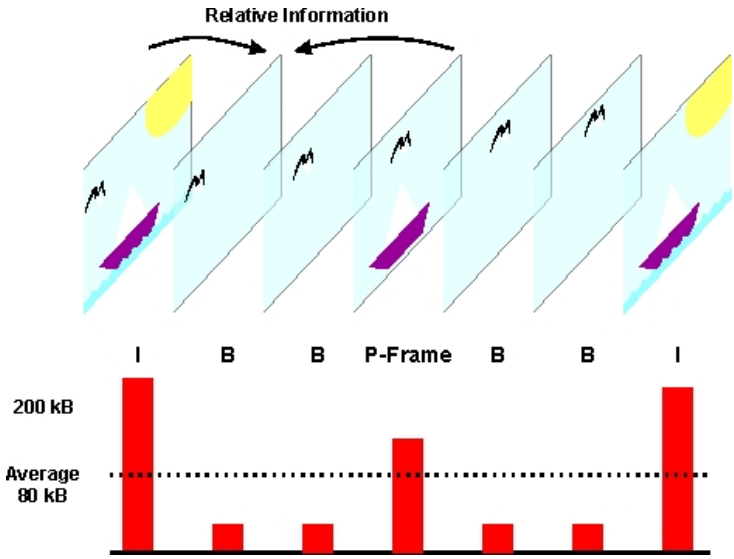


Figure 2.9: Illustration showing how a b-frame is constructed

camera. With the short introduction above to the different types of video frames and how they are dependent on each other, one could see that this becomes a problem. If you want to decode frame N and frame N is not an I-frame, you have to find the last I-frame, and then decode every frame up to frame N before you can decode frame N.

In this thesis, we have used x264 – a free software library for encoding video streams into the H.264/MPEG-4 AVC format. The standard has almost endless possibilities in terms of parameters and adjustments, however it could be broken down to three parameters. Which profile it should use, amount of compressions and encoding speed, each affecting each other. The profile determines which features in the standard that should be utilized and the standard defines 21 sets of capabilities, targeting specific classes of applications.

### 2.2.5 Synchronizing frames

Previously in this chapter, we introduced the camera controller and the trigger box that synchronizes each camera's shutter. Later in this thesis we will look at how we can stitch video using multiple cameras and specifically we will see how we can combine frames from four cameras into one large panoramic view. Video stitching puts another demand on our video recording software and that is synchronization between cameras and captured frames.

Each camera is controlled independantly by a seperate computer and even tho the cameras have synchronized shutters, the recorded frame does not contain any information on when it was recorded so we needed some kind of synchronization of frames. Using system clock timestamps

was out of the question since millisecond precision is needed and we did not have a reliable Internet connection to utilize Network Time Protocol (NTP).

Our first approach was creating a synchronization point in the video, and count frames from this synchronization point. We did some lab testing using red LEDs in the camera housing and detecting when they were lit in our software to create a synchronization point. This worked fine in an indoor environment, however under outdoor lighting conditions, we were unable to detect when the LEDs were lit. This led a more robust solution developed by Alexander Eichhorn as part of the Northlight library.

### 2.2.6 TimeCodeServer

In order to keep video synchronized, we needed a way to synchronize video frames over several machines connected by an IP network, in a robust and efficient way. This led to the development of TimeCodeServer.

TimeCodeServer is an Open Sound Control (OSC) based server for time code synchronization. OSC is a content format for messaging among computers, sound synthesizers, and other multimedia devices that are optimized for modern networking technology. The main goal for TimeCodeServer is to establish consensus between all clients in a distributed camera array. Our implementation is based on the well known Berkley algorithm described in [22]. However, in contrast to the Berkely protocol which is intended to synchronize system clocks, we synchronize a media clock implemented on top of the system clock since we cannot assume to have sufficient priviledges to adjust system time. Another difference is that we do not use ICMP Timestamp request/reply messages, but instead build our own protocol using Open Sound Control message format and UDP messages at application level. A third difference is that we also exchange a lower granularity timecode. A monitor application was made to monitor the server and can be seen in figure 2.10.

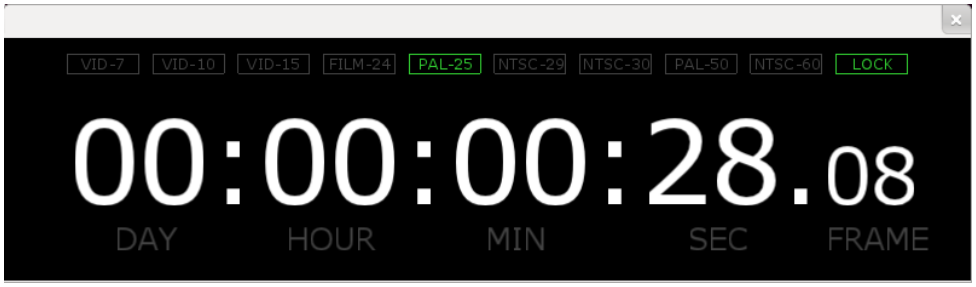


Figure 2.10: Timeserver monitor where server runs a PAL-25 compatible timecode (25 frames per second)

The timecode server decides about intervals in which individual clients get contacted depending on the reliability of knowledge on client clock drift. New clients and clients with more variability in local clock drift get contacted more often, but at least once a minute.

Clients are supposed to immediately answer server requests. On failure to do so the server tries contacting clients more frequently. After a specified time without answer the server assumes client failure and removes soft-state.

**Algorithm Objectives**

1. Timecode increases strictly monotonic.
2. Each timecode is allocated to at most one video frame per camera.
3. Video frames from different cameras captured during the same global.
4. Time interval at different nodes get the same timecode.

**Algorithm assumptions**

1. Message channels are unreliable bi-directional unicast (e.g. IP network).
2. Message sending latency is equal in both directions (Client <-> Server).
3. Local clocks may drift and camera shutter may jitter.
4. Clock drift is slow.
5. Message processing is subject to CPU scheduling (no in-kernel replies).
6. Local system time may or may not be synchronised (e.g. by NTPv4).

In order to achieve those objectives, the algorithm controls three client parameters:

**Timecode Value**

New clients must obtain an initial timecode value before valid timecodes can be assigned to camera frames. After that, the algorithm adjusts timecode values of clients continuously, but it is up to the client to decide about frame drop, frame skip or whether it leaves the stream out of sync.

**Timecode Interval Length**

At each individual client the algorithm adjusts interval length of timecodes in units of nanoseconds relative to the local system clock. That is, the duration of each timecode interval as measured in local system time may differ between nodes due to differences in

local clock speed. Knowing the interval length, clients can decide independently when to advance their local timecodes regardless of time server availability.

**Timecode Interval Phase**

For autonomous timecode allocation it is also necessary that each client can compute the time (again in local clock ticks) at which to advance its timecode. The server sends an adjustment value for the phase parameter so that all clients use the same phase. The optimal phase (i.e. start of a timecodes interval is exactly between two frame capturing events. Thus, camera jitter can be compensated best.

**Remaining synchronization error**

An uncertainty window in camera frame timestamping still remains due to inaccuracy of message latency measurement and local camera jitter. Clients should therefore compensate for this by extending the timecode interval by a window of size equal to message RTT jitter and camera jitter on both sides of the interval. Uncertainty windows of subsequent intervals shall overlap. Then, timecodes should be allocated to any two subsequently captured frames according to the following rules:

Rule 1

If any two subsequently captured video frames are captured within the same extended interval, then the first frame receives the timecode of the interval and the second frame receives the subsequent timecode only if it has been captured during the late extension limit. Otherwise the second frame must be dropped.

Rule 2

If no frame has been captured within the extended interval, then one timecode is skipped.

## 2.3 Open source computer vision library

Open Source Computer Vision Library (OpenCV) [18] is – as it name states – an open source computer vision library. It includes several hundred algorithms in the areas of image processing and computer vision. It contains algorithms for image filtering, geometrical image transformations (resize, affine and perspective warping), video analysis – including motion estimation and object tracking, basic multiple view geometry, single and stereo calibration, feature detectors, feature descriptor and descriptor matching. Many of which we will use in this thesis, the first being the camera calibration module. Removing barrel distortion involves moving or remap-

ping pixels, and before we look at how we can remove barrel distortion, we will look at some effects that can occur when remapping images.

### 2.3.1 Image interpolation

Image interpolation occurs in almost all digital photos at some stage. It happens anytime you resize or remap (distort) your image from one pixel grid to another. Image resizing is necessary when you need to increase or decrease the total number of pixels, whereas remapping can occur under a wider variety of scenarios: correcting for lens distortion, changing perspective, and rotating an image.

Even if the same image resize or remap is performed, the results can vary significantly depending on the interpolation algorithm. It is only an approximation, therefore an image will always lose some quality each time interpolation is performed.

Interpolation works by using known data to estimate values at unknown points. For example, if you wanted to know the temperature at noon, but only measured it at 11:00 and 13:00, you could estimate its value by performing a linear interpolation. Image interpolation works in two directions, and tries to achieve a best approximation of a pixel's color and intensity based on the values of the surrounding pixels and figure 2.11 on the next page illustrates this, using bilinear interpolation.

Common interpolation algorithms can be grouped into two categories – adaptive and non-adaptive. Adaptive methods change depending on what they are interpolating (sharp edges vs. smooth texture), whereas non-adaptive methods treat all pixels equally.

Non-adaptive algorithms include nearest neighbor, bilinear, bicubic, spline, sinc, lanczos and others. Depending on their complexity, these use anywhere from 0 to 256 (or more) adjacent pixels when interpolating. The more adjacent pixels they include, the more accurate they can become, but this comes at the expense processing time. These algorithms can be used to both distort and resize an image, both of which are considered in this thesis.

OpenCV has support for the following interpolation algorithms, listed with increasing complexity: nearest neighbor, bilinear, bicubic and lanczos interpolation. In image processing, bicubic interpolation is often chosen over bilinear interpolation or nearest neighbor in image resampling, when speed is not an issue. In contrast to bilinear interpolation, which only takes 4 pixels (2x2) into account, bicubic interpolation considers 16 pixels (4x4). Images resampled with bicubic interpolation are smoother and have fewer interpolation artifacts. Lanczos has

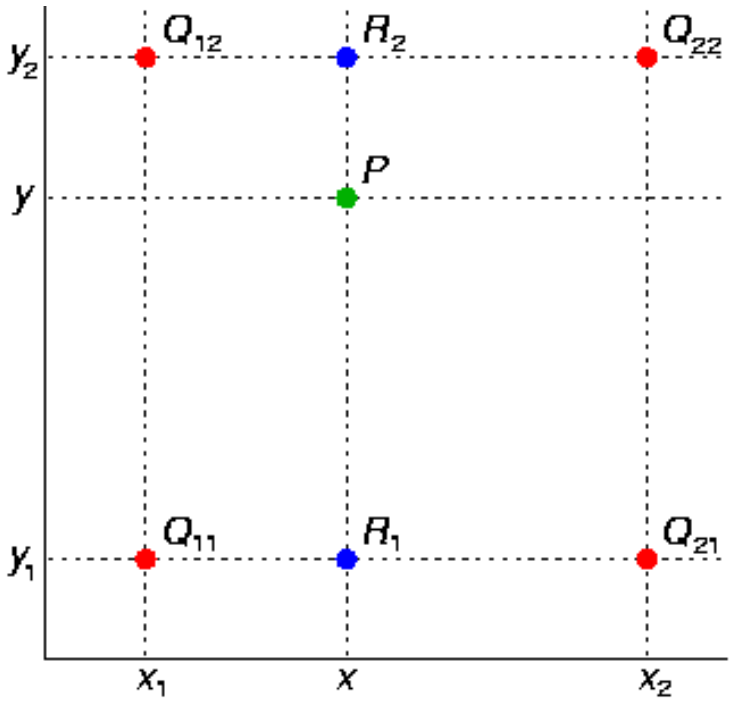


Figure 2.11: The four red dots show the data points and the green dot is the point at which we want to interpolate.

| | Min | Max | Mean |
|---|---|---|---|
| Nearest neighbor interpolation | 4.14498 ms | 5.08296 ms | 4.21125 ms |
| Bilinear interpolation | 7.39556 ms | 7.8408 ms | 7.44553 ms |
| Bicubic interpolation | 47.8951 ms | 55.3249 ms | 48.2764 ms |
| Lanczoz interpolation | 240.129 ms | 245.435 ms | 240.664 ms |

Table 2.1: A speed comparison between different interpolation algorithms when remapping 300 frames.

the advantages of bicubic and is known to produce sharper results than bicubic interpolation. Table 2.1 show a speed comparison between the algorithms implemented in OpenCV.

### 2.3.2 Camera calibration

Camera calibration may refer to photometric camera calibration and geometric camera calibration. While an interesting area, photometric camera calibration is not in the scope of this thesis and we will only consider geometric camera calibration.

Geometric camera calibration – or camera resectioning, is the process of finding the true parameters of the camera that produced a given image. The pinhole camera model (see figure 2.12 on the next page) describes the mathematical relationship between the coordinates of a three-

dimensional point and its projection onto the image plane of an *ideal* pinhole camera, where the camera aperture is described as a point and no lenses are used to focus light. The model does not include, for example, geometric distortions or blurring of unfocused objects caused by lenses and finite sized apertures. It also does not take into account that most practical cameras have only discrete image coordinates. This means that the pinhole camera model can only be used as a first order approximation of the mapping from a 3D scene to a 2D image. Its validity depends on the quality of the camera and, in general, decreases from the center of the image to the edges as lens distortion effects increase.

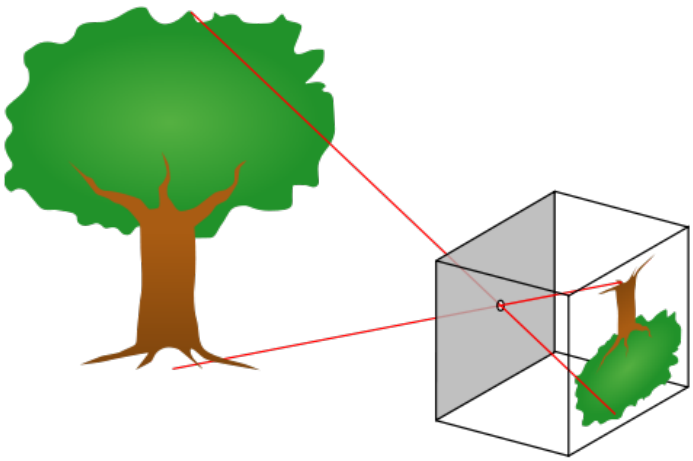

Figure 2.12: A diagram of the pinhole camera model

In our prototype system, we use cheap industrial cameras and more importantly we use cheap optics for our cameras. Also, since our prototype setup only consist of four cameras, we needed wide angel lenses with fairly short focal length in order to cover the entire soccer field. It is known that cheap optics suffers from significantly more *optical aberrations* compared to more expensive optics due to the lower quality of the glass used in the lens elements. An *optical aberration* is a departure of the performance of an optical system and there are many types of optical aberrations, however we will only consider image distortion in this thesis. Wide angel lenses often suffers from barrel distortion. Barrel distortion is the effect where the center of the image is magnified more than the perimeter (see figure 2.13 on the following page).

There exist many algorithms and techniques for correcting barrel distortion and some requires some kind of reference grid while other are automatic [23]. The removal of barrel distortion is a well studied subject in the field of computer vision and we will not be going into the details of it in this thesis. However, the basics is to find a mathematical model that models the distortion, and remap the pixels according to this model to make an undistorted image.

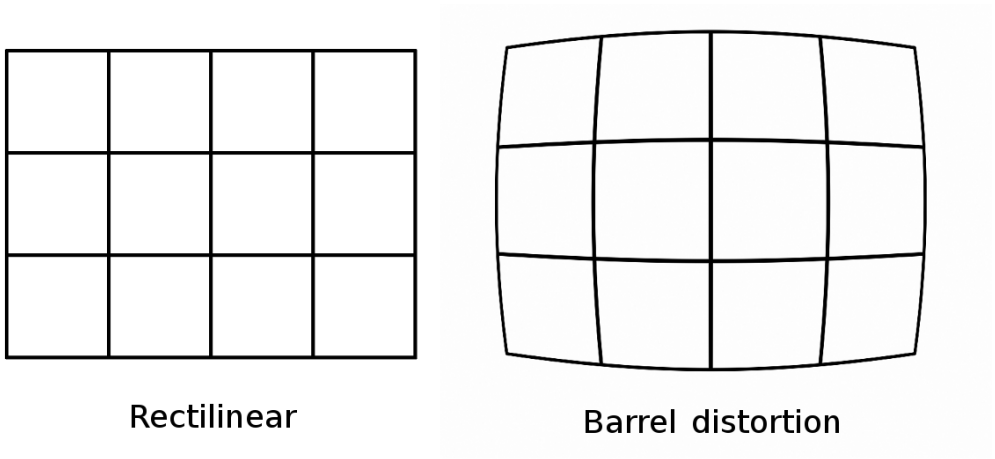


Figure 2.13: Illustration of barrel distortion

We have used OpenCV in this thesis to remove barrel distortion and OpenCV uses a technique where you take an image of a reference grid to calculate the mathematical model of the distortion. Without going into to much detail about image distortion, OpenCV corrects two types of distortion. *Radial distortion*, which manifests itself in the form of "barrel" or "fisheye" effect and *tangential distortion*, which is an effect that occurs because the image taking lens is not perfectly parallel to the imaging plane. The process of removing image distortion in OpenCV consist of the following steps:

**Compute the mathematical model using reference images**

This step is a step that we only need to do once and it can be seen as a setup or configuration step. This step takes a series of images of the same reference board as input, and the output is the distortion coefficients in the mathematical model. We have used a chessboard pattern, where the corners between the black and white squares are used as reference points. Figure 2.15 on the next page shows this chessboard pattern where the detected corners are drawn on the image.

**Build look up table**

To debarrel or rectify the image, each pixel in the input image must be moved to a specific output pixel. Since our lenses do not have any optical zoom, this relationship is constant for every frame in the whole video stream. The calculations for a single pixel can be seen in figure 2.14 on the facing page and this computation is the same for every pixel (x,y) in subsequent video frames. To save computation time, a lookup table is created for when the first frame is debarreled.

**Debarreling**

When the lookup table is built, we can do the remapping of the input pixels using one of

$$
\begin{aligned}
x &\leftarrow (u - C'_x)/f'_x \\
y &\leftarrow (v - c'_x)/f'_y \\
\begin{bmatrix} X & Y & W \end{bmatrix}^T &\leftarrow R^{-1} * \begin{bmatrix} x & y & 1 \end{bmatrix}^T \\
x' &\leftarrow X/W \\
y' &\leftarrow Y/W \\
x'' &\leftarrow x'(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + 2p_1 x' y' + p_2(r + 2x'^2) \\
y'' &\leftarrow y'(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + p_1(r^2 + 2y'^2) + 2p_2 x' y' \\
map_x(u, v) &\leftarrow x'' f_x + c_x \\
map_y(u, v) &\leftarrow y'' f_y + c_y
\end{aligned}
$$

Figure 2.14: Calculations for every input pixel (x,y) to every output pixel (u,v). The meaning of many symbols are left and the interested reader can look at the OpenCV documentation [18]

the interpolation algorithms described in section 2.3.1.

$$output(x, y) = input(map_x(x, y), map_y(x, y))$$

In figure 2.16 on the next page, we can see the chessboard image debarreled.

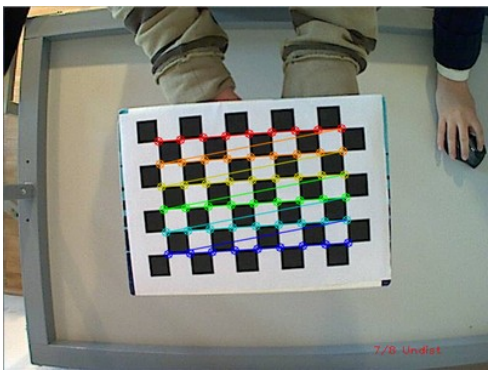

Figure 2.15: An image of the chess board where the corners are detected [18]

## 2.4 Evaluation and discussion

### 2.4.1 Video synchronization

In this chapter we have introduces a robust method for synchronizing multiple camera streams by assigning synchronized time codes to each video frame. With the underlying assumptions, our method has proven to be successful under our lab tests. Compared to our first approach where we used synchronization points in the video, using a server/client approach to distribute

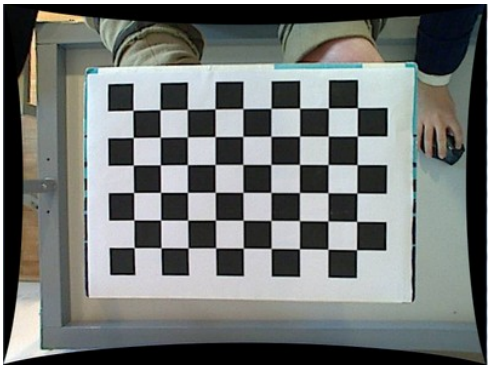

Figure 2.16: An image of the same chess board where the distortion is removed [18]

timestamps, enables us to integrate the frame synchronization with other components such as the sensors network introduced in the next chapter. The main problems with using timestamps to synchronize video frames is the lack of support for frame-by-frame metadata in video encoders.

### 2.4.2 Dropped frames

Frame drops can be caused by a number of events in our system. In our prototype, video is recorded at 30 frames per second, which means that each frame only has $1/30second$ to do any processing and keep the system from dropping frames. If more than $1/30second$ is spent, the next frame will be dropped. Frame drops can also happen, if the TimeCodeServer fails to acquire a timecode for a frame or the camera itself can drop frames. It is important to handle dropped frames in order to keep streams synchronized. When a frame is dropped, you are basically forced to do one of two things in order to keep streams synchronized. One, insert a black or empty frame, or two – insert the previous recorded frame. When frame rates are as high as in our prototype, the viewer hardly notices if a the same frame is displayed two times, since there are very little changes at these frame rates.

### 2.4.3 Debarreling

In table 2.1, we showed a speed comparison of remapping pixels using different interpolation algorithms. As we can see in the table, bicubic- and lanczos-interpolation is to slow to keep the recorder from dropping frames. It is worth mentioning that the performance measurements are done using a single thread, and interpolation is an embarrassingly parallel problem, i.e. each pixels computation is independent of the other pixels. When recording it is better to keep the data as close to the original as possible, and rather do processing on the recorded data.

This is why we have left debarreling out of the recorder and rather implemented it into viewer application.

### 2.4.4 Encoding

Each video frame in our system is synchronized using timestamps. In order to keep streams synchronized, this metadata has to integrated into the stored stream. While the JPEG format has support for metadata, the H.264 standard does not have this support. In order to keep this metadata, and be extension keep streams synchronized, some kind of container format is needed.

Another possible approach is storing each encoded frame as a separate file, where the timestamp is used in the filename. However, if a video encoder like x264 is used, it puts restrictions on which frame types that can be used. Or alternatively, you got to keep track of which files are of which frame type in order to decode them properly.

In our scenario, where a viewer may switch video streams at any given time, there is one disadvantage with using a video encoder like x264. Depending on the frame types used in the encoding, in order to playback synchronized streams, all streams has to be decoded even though we are only interested in only one of them.

In the next chapter, we introduce an approach for integrating a positioning system with video in order to track and follow players. One could imagine a scenario where your watching a game and you are following a player and the player is sprinting over the entire field. At the start of the stream (frame 0), the player is located in one camera. And after ten seconds (frame 300) the player is located in another camera. Both camera streams are encoding using x264, where frame 0 in both streams are I-frames and the rest are P-frames. In order to switch streams seamlessly and without any loading or pauses, each frame both streams has to be decoded even tho we are only watching one of the streams at the time. This is because frame 300 is a P-frame and decoding a P-frame relies on decoding the previous frame. So in order to decode frame 300, all frames from 0 to 299 has to be decoded.

In table 2.2 on the following page, a comparison between different compression techniques is given. Notice, that P-frames is turned off for some of the x264 encoding results.

In terms of encoding, it really does come down to a trade off between encoding speed, decoding speed and disk requirements and choosing the appropriate format. If disk space is not an issue, video encoding with only I-frames might be the best option, as this relieves the playback

| Compression method | Average frame size | Storage size (300 frames) | Encoding Speed (fps) | I-frames | P-frames |
|---|---|---|---|---|---|
| YUV 4:2:0 (no compression) | 1786 kB | 536 MB | $\infty$ | - | - |
| Lossy JPEG | 243 kB | 73 MB | 42 | - | - |
| Lossy x264 (normal preset) | 47 kB | 14 MB | 113 | 300 | 0 |
| Lossless x264 (normal preset) | 753 kB | 226 MB | 34 | 300 | 0 |
| Lossless x264 (slow preset) | 490 kB | 147 MB | 3 | 2 | 298 |
| Lossless x264 (ultrafast preset) | 583 kB | 175 MB | 99 | 2 | 298 |

Table 2.2: A comparison between image compression and video encoding sizes and encoding time when encoding 300 frames in a stream.

software from decoding multiple streams. But as we showed in table 2.2, using only I-frames has a severe impact on encoded frame size.

The future for encoding synchronized video of the same scene is *Multiview Video Coding (MVC)*. MVC is an amendment to the H.264/MPEG-4 AVC video compression standard which enables efficient encoding of sequences captured simultaneously from multiple cameras. Even tho the standard was finalized in June 2009, there are still few or none free encoders who supports it.

### 2.4.5 Camera calibration

In section 2.3.2, we introduced a method for geometrically calibrating cameras. In our prototype on Alfeim stadium, we were unable to record checkerboard to use for calibration and instead we calibrated the cameras in our lab. Even tho we used the same optics and cameras, they produced slightly different results which led to small calibration errors. As we can see in figure 2.17, some lines which should have been straight have a slight bend because the barrel distortion is not completely removed. This is especially true for the line between the corner markers.

## 2.5 Summary

In this chapter we have looked at several aspects in regards to capturing video from multiple sources and keeping them synchronized and we have highlighted and identified problem areas when using general purpose hardware and industry standard components. We have also looked


Figure 2.17: The image shows a debarreled image where red straight lines are drawn to illustrate that there is still some barrel distortion in the image.

at how we can rectify a distorted video stream in real-time, however we concluded that it should not be part of recording software, but rather be post-processed.

Now that we have introduced a method to keeping multiple streams synchronized, we will investigate how can use these streams in combination with a sensor-network in order to track players in the streams.

# Chapter 3

# Player Tracking

In the previous chapter, we looked at how we could capture and synchronize multiple cameras. In this chapter we will look at how we can track an object through multiple synchronized cameras using a high precision sensor network. By tracking, we mean the process of locating an object and also identifying which camera(s) the object is in. But first, we will have a look at our sensor network.

## 3.1 ZXY

ZXY Sport Tracking AS (ZXY) [5] is a commercial company delivering sport tracking solutions to, amongst others, Tromsø IL. Their solutions are based on wireless technology using the 2.45 GHz (and optionally 5.2 GHz) spectrum. According to themselves, their solutions are capable of tracking their transponders with centimeter accuracy, at a rate of up to 40 samples per second [5]. However, the solution currently installed at Tromsø is somewhat older and delivers 20 samples per second, with a accuracy of one meter – depending on the location on the field. Figure 3.1 on the next page shows an illustration of ZXYs system, while figure 3.2 on the following page shows one of the antennas currently installed at Alfeim Stadium.

### 3.1.1 ZXY Data

The sensors record each players position at a rate of 20 Hz, which are processed and stored in a database. While the primary function of the sensor system is tracking the position of each player or transponder, ZXY's system have aggregated lots of additional data from the positioning data.


Figure 3.1: Illustration image of ZXY's sensor technology [5]


Figure 3.2: One of ZXY's antennas at Alfeim Stadion

Figure 3.3 on the next page shows some of the raw data and its format in ZXYs database. As we can see, they have aggregated data like speed, heading and the sensors also records step frequency. While many of these datas are interesting in a video tracking scenario, our primary focus has been the actual positions of the players and how we can utilize them in a video scenario. For instance, the system automatically annotates some events like corners and penalty kicks based on player positions. In a later chapter, we will see how we can utilize these additional datas.

While the positioning data is recorded automatically, there are also some data which requires manual input. Each transponder has its own unique id to separate them and the link between player name and transponder id has to be set up for each session. There are also some events that are recorded manually for each session or game, and the most important ones are "game

start" and "period start". This is especially important for synchronizing video and sensor data.

### 3.1.2 Synchronizing video and positioning data

As mentioned the ZXY system records a time stamp which indicates the kickoff of each period. This time stamp is very important and later in this thesis, we will introduce a prototype which integrates the positioning system with our video capture software. Since we did not have full access to ZXY's system, we were unable to implement some kind of time synchronization between our recorder and the ZXY database. However, we have already showed an implementation of time synchronization (see section 2.2.6 on page 18) between multiple cameras, and this could be integrated into the ZXY system. In our prototype, the synchronization between the sensor network timestamps and the video timestamps has been done manually by matching the video timestamp with the sensor network timestamp.

| | timestamp | tag_id | x_pos_grid | y_pos_grid | x_pos | y_pos | restarted_oversample | speed | total_distance | heading | steps | direction | mac_indication | step_frequency |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2011-11-20 17:47:18.462 | 19 | 75 | 34 | 75.5188140869 | 33.8997344971 | 0 | 1.764957428 | 4.5921397209 | 0.2723002732 | 0 | -0.3632633686 | 74 | 1.4302062988 |
| 2 | 2011-11-20 17:47:18.512 | 19 | 75 | 34 | 75.485168457 | 33.9831123352 | 0 | 1.8100215197 | 4.6820516586 | 0.2549702227 | 0 | -0.3896487057 | 74 | 1.5211874247 |
| 3 | 2011-11-20 17:47:18.562 | 19 | 75 | 34 | 75.4505844116 | 34.0622444153 | 0 | 1.6990865469 | 4.7684140205 | 0.1929336339 | 8 | -0.4207410216 | 74 | 1.5842279196 |
| 4 | 2011-11-20 17:47:18.611 | 32 | 69 | 15 | 69.4959182739 | 15.5709009171 | 0 | 1.976149559 | 4.698539257 | -1.1793929338 | 0 | -1.5261707306 | 72 | 1.5068434477 |
| 5 | 2011-11-20 17:47:18.612 | 19 | 75 | 34 | 75.413772583 | 34.1375007629 | 0 | 1.6690102816 | 4.8521976471 | 0.1015008464 | 0 | -0.4673682451 | 74 | 1.6388835907 |
| 6 | 2011-11-20 17:47:18.661 | 32 | 68 | 15 | 69.3978347778 | 15.5751037598 | 0 | 1.9591518641 | 4.7967128754 | -1.1727112532 | 0 | -1.5284729004 | 72 | 1.5476746559 |
| 7 | 2011-11-20 17:47:18.662 | 19 | 75 | 34 | 75.3753128052 | 34.2120056152 | 0 | 1.6808278561 | 4.9360442162 | 0.002661135 | 0 | -0.4801314175 | 74 | 1.6819070578 |
| 8 | 2011-11-20 17:47:18.711 | 32 | 68 | 15 | 69.3005142212 | 15.5790739059 | 0 | 1.9443223476 | 4.8941140175 | -1.164116025 | 0 | -1.5305651426 | 72 | 1.5710529089 |
| 9 | 2011-11-20 17:47:18.712 | 19 | 75 | 34 | 75.3368530273 | 34.283416748 | 0 | 1.6020314693 | 5.0171546936 | -0.0878919438 | 0 | -0.4989666641 | 74 | 1.7064301968 |
| 10 | 2011-11-20 17:47:18.761 | 32 | 68 | 15 | 69.2063522339 | 15.5828561783 | 0 | 1.8633719683 | 4.9883518219 | -1.1551482677 | 8 | -1.5307424068 | 72 | 1.58132267 |
| 11 | 2011-11-20 17:47:18.762 | 19 | 75 | 34 | 75.2971038818 | 34.3499794006 | 0 | 1.5345220566 | 5.0946893692 | -0.1616109014 | 0 | -0.5518113375 | 74 | 1.7735360861 |
| 12 | 2011-11-20 17:47:18.811 | 32 | 68 | 15 | 69.113243103 | 15.5863361359 | 0 | 1.8643434048 | 5.0815258026 | -1.1475439072 | 0 | -1.5342441797 | 72 | 1.5892320871 |
| 13 | 2011-11-20 17:47:18.812 | 19 | 75 | 34 | 75.2564468384 | 34.4130210876 | 0 | 1.488743186 | 5.1697063446 | -0.2139298916 | 0 | -0.5803841352 | 74 | 1.848395586 |
| 14 | 2011-11-20 17:47:18.861 | 32 | 68 | 15 | 69.0198669434 | 15.5894804001 | 0 | 1.8714241982 | 5.174955368 | -1.1446191072 | 0 | -1.538074255 | 72 | 1.5999507904 |
| 15 | 2011-11-20 17:47:18.862 | 19 | 74 | 34 | 75.2150421143 | 34.4727020264 | 0 | 1.4395449162 | 5.2423462868 | -0.2417210042 | 0 | -0.6151112318 | 74 | 1.8619407415 |
| 16 | 2011-11-20 17:47:18.911 | 32 | 68 | 15 | 68.9284362793 | 15.5919923782 | 0 | 1.8157908916 | 5.2664208412 | -1.1446191072 | 0 | -1.5449740887 | 72 | 1.6058236361 |
| 17 | 2011-11-20 17:47:18.912 | 19 | 74 | 34 | 75.1715774536 | 34.5294151306 | 0 | 1.4258749485 | 5.3138055801 | -0.242127493 | 0 | -0.6671532989 | 74 | 1.8739501238 |
| 18 | 2011-11-20 17:47:18.961 | 32 | 68 | 15 | 68.8380889893 | 15.5941905975 | 0 | 1.8031822443 | 5.3567948341 | -1.1446191072 | 0 | -1.5470366478 | 72 | 1.614710927 |
| 19 | 2011-11-20 17:47:18.962 | 19 | 74 | 34 | 75.1279678345 | 34.5809822083 | 0 | 1.3251390457 | 5.3813457489 | -0.231394887 | 4 | -0.7147462368 | 74 | 1.8893432617 |
| 20 | 2011-11-20 17:47:19.011 | 32 | 68 | 15 | 68.7484207153 | 15.596159935 | 0 | 1.7911801338 | 5.4464845657 | -1.1446191072 | 0 | -1.5494337082 | 72 | 1.6614705324 |
| 21 | 2011-11-20 17:47:19.012 | 19 | 74 | 34 | 75.0834350586 | 34.628112793 | 0 | 1.2874386311 | 5.4461936951 | -0.231394887 | 0 | -0.7710667253 | 74 | 1.9043040276 |
| 22 | 2011-11-20 17:47:19.061 | 32 | 68 | 15 | 68.6597061157 | 15.5982561111 | 0 | 1.7693277597 | 5.5352239609 | -1.1446191072 | 0 | -1.5463446379 | 72 | 1.723574996 |
| 23 | 2011-11-20 17:47:19.062 | 19 | 74 | 34 | 75.038230896 | 34.6687812805 | 0 | 1.1937208176 | 5.5070180893 | -0.231394887 | 0 | -0.8633901477 | 74 | 1.9159750938 |
| 24 | 2011-11-20 17:47:19.062 | 25 | 85 | 34 | 85.1776428223 | 34.1191482544 | 0 | 2.3321251869 | 4.0015969276 | -0.9247873425 | 0 | 1.6960927248 | 42 | 5.8706202507 |
| 25 | 2011-11-20 17:47:19.111 | 32 | 68 | 15 | 68.5719299316 | 15.6002693176 | 0 | 1.7527598143 | 5.6230230331 | -1.1369583607 | 0 | -1.5483776331 | 72 | 1.7537989616 |
| 26 | 2011-11-20 17:47:19.112 | 19 | 74 | 34 | 74.9922409058 | 34.702999115 | 0 | 1.1329779625 | 5.5643587112 | -0.231394887 | 0 | -0.9561204314 | 74 | 1.9666638374 |
| 27 | 2011-11-20 17:47:19.112 | 25 | 85 | 34 | 85.2935638428 | 34.1032028198 | 0 | 2.3446879387 | 4.1186103821 | -0.9247873425 | 0 | 1.7116349936 | 42 | 6.1170291901 |
| 28 | 2011-11-20 17:47:19.161 | 32 | 68 | 15 | 68.4850082397 | 15.6020889282 | 0 | 1.7337598801 | 5.7099637985 | -1.1266468763 | 4 | -1.5503766537 | 72 | 1.8006689548 |
| 29 | 2011-11-20 17:47:19.162 | 19 | 74 | 34 | 74.9454574585 | 34.7320594788 | 0 | 1.0920600891 | 5.6194448471 | -0.231394887 | 10 | -1.0361039639 | 74 | 2.0696349144 |
| 30 | 2011-11-20 17:47:19.162 | 25 | 85 | 34 | 85.4090118408 | 34.0880508423 | 0 | 2.32452178 | 4.2350492477 | -0.9247873425 | 0 | 1.6974310875 | 42 | 6.2101178169 |
| 31 | 2011-11-20 17:47:19.211 | 32 | 67 | 15 | 68.3991775513 | 15.603723526 | 0 | 1.7120251656 | 5.7958097458 | -1.1134384871 | 0 | -1.5522568226 | 72 | 1.8578704596 |
| 32 | 2011-11-20 17:47:19.212 | 19 | 74 | 34 | 74.8970565796 | 34.7542304993 | 0 | 1.0595731735 | 5.6727180481 | -0.231394887 | 0 | -1.1783093214 | 74 | 2.1427836418 |
| 33 | 2011-11-20 17:47:19.212 | 25 | 86 | 34 | 85.5231628418 | 34.0749206543 | 0 | 2.289513588 | 4.3499541283 | -0.9247873425 | 0 | 1.6807175875 | 42 | 6.2584519386 |
| 34 | 2011-11-20 17:47:19.261 | 32 | 67 | 15 | 68.3145294189 | 15.605173111 | 0 | 1.6875510216 | 5.8804702759 | -1.0965162516 | 0 | -1.5541356802 | 72 | 1.9166890383 |
| 35 | 2011-11-20 17:47:19.262 | 19 | 74 | 34 | 74.8490447998 | 34.7707481384 | 0 | 1.0001392365 | 5.7235016823 | -0.231394887 | 0 | -1.2597565651 | 74 | 2.1580679417 |
| 36 | 2011-11-20 17:47:19.262 | 25 | 86 | 33 | 85.6369552612 | 34.0640869141 | 0 | 2.2851805687 | 4.4642624855 | -0.9247873425 | 0 | 1.660458684 | 42 | 6.4898233414 |
| 37 | 2011-11-20 17:47:19.311 | 32 | 67 | 15 | 68.2312088013 | 15.6064109802 | 0 | 1.6594103575 | 5.9637999535 | -1.0755919218 | 0 | -1.556497097 | 72 | 1.9769277641 |

Results Messages

Line 1 Column 47 100 rows

Figure 3.3: Raw data from ZXYs database

### 3.1.3 Coordinate system

Soccer fields does not have a standard size (width and length) and Alfheim stadium has a field measuring $105meters \times 68meters$ and ZXY's coordinate system uses these lengths (see figure 3.4 on the next page), where coordinate (0,0) is located in the lower left corner (from the cameras point of view) and (105,68) is located in the upper right corner. Since ZXY's system uses the real sizes as coordinates, we are able to derive the coordinates of center field, penalty

marker and other important markers using figure 3.4. Later, we will see why it is important to know the exact coordinates of these markers.

Now that we have seen the how the ZXY sensor system works and we know its format, we will see how we can utilize such a system in combination with video.

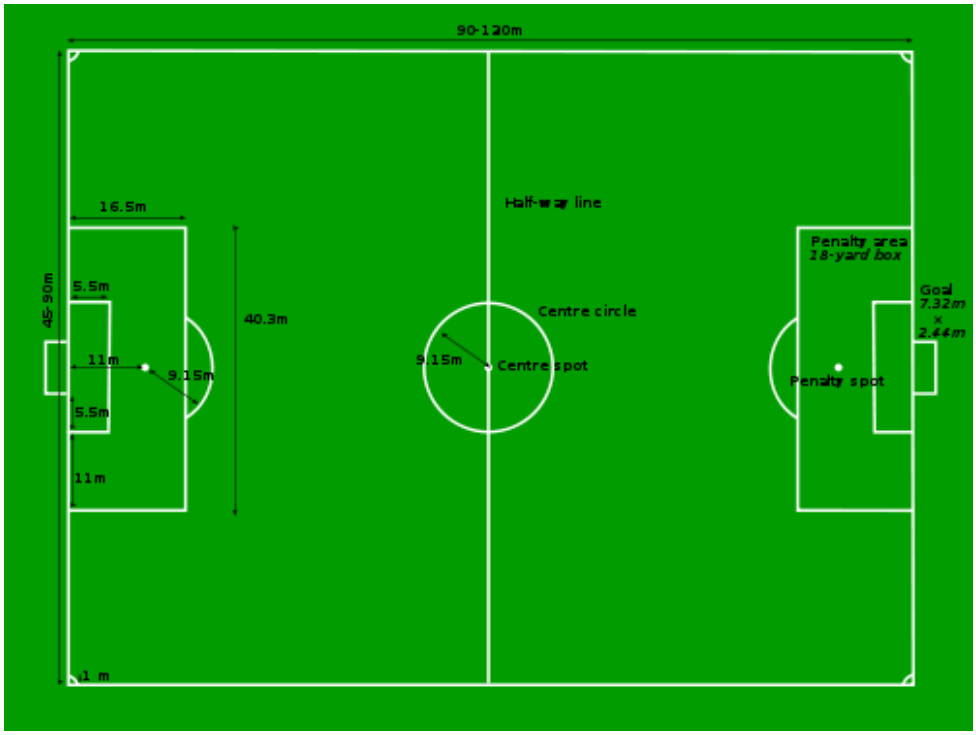


Figure 3.4: Standard soccer field measurements.

## 3.2 Following a player

As our first scenario, we wanted to follow a player by switching camera according to the players position on the field. This is the most lightweight solution in terms of computation power, and it is not player tracking by our definition in this thesis. This simple solution checks the position of the player we are following, and then selects the camera the player is located at. To be able to find the correct cameras, some predefined boundaries was made to define which areas of the field each camera covers. Our tests showed that in our setup, it was sufficient to create borders along the X-axis and ignore the Y-axis of the field since our cameras have quite a lot of overlap. Figure 3.5 on the next page shows an implementation of this camera selector.

While this solution gave us the ability to follow a player using camera switching without manual intervention , we wanted a higher precision in our tracking and we wanted the exact position of

the players.

```
int ZXYCoordinateHelper::selectCamera ( double x, double y ) {
    int i;

    if( x < 35) {
      return 0;  //First camera
    }  else if(x < 52.5) {
      return 1;  //Second camera
    } else if(x < 75) {
      return 2;  //Third camera
    } else {
      return 3;  //Fourth camera
    }
    return -1; //Error
}
```

Figure 3.5: Function that finds the optimal camera for a player by predefined boundaries

## 3.3 High precision tracking

Video tracking is the process of locating an object in a video stream. We wanted to track and identify each player on the field. This is possible to do without the use of sensor networks, however the process of tracking objects in video is computationally expensive. Adding to this complexity is object recognition if we want to identify who we are tracking. We will investigate how we can utilize a sensor network to remove the computational complexity. In regards to high precision tracking, we wanted to find out the exact pixel coordinates in a video frame a player is located at, and additionally which camera(s) he is in.

### 3.3.1 Projective geometry

As already mentioned, each player wears a unique transponder which records each players position on the field. The position is a coordinate in the Cartesian coordinate system. In order to locate a player in the video, we need a translation from $ZXY(x, y) \rightarrow pixel(u, v)$ which is correct, for all valid pixel coordinates in a video frame.

A translation from coordinate system $ZXY(x, y) \rightarrow pixel(u, v)$ can be represented by a *transformation matrix*. In our case, we need a transformation called a perspective transformation. In

the scientific literature, synonyms like collineation, homography and projectivity are often used for perspective transformations.

Formally, a projective transformation in a plane is a transformation used in projective geometry. It is the composition of a pair of perspective projections. It describes what happens to the perceived positions of observed objects when the point of view of the observer changes. Projective transformations do not preserve sizes or angles (in contrast to affine transformations), but do preserve incidence and cross-ratio: two properties which are important in projective geometry. In the field of computer vision, any two images of the same planar surface in space are related by a homography (assuming a pinhole camera model). This has many practical applications, such as image rectification, image registration, or computation of *camera motion*, i.e. rotation and translation between two images. Once camera rotation and translation have been extracted from an estimated homography matrix, this information may be used for navigation, or to insert models of 3D objects into an image or video, so that they are rendered with the correct perspective and appear to have been part of the original scene. A projective transformation for a line is illustrated in figure 3.6.

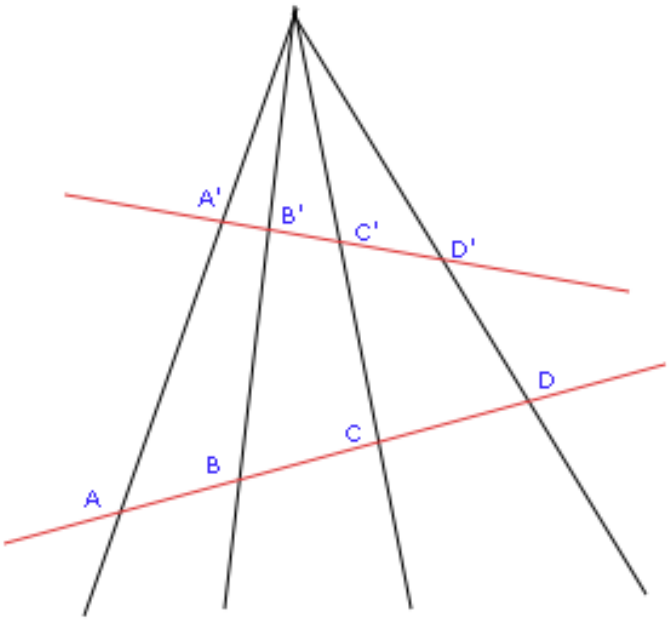


Figure 3.6: Points A, B, C, D and A', B', C', D' are related by a projective transformation.

A projective transformation can be expressed by a 3x3 matrix (see figure 3.8 on the next page) and the translation from one coordinate system to another is calculated as described in 3.7 on the facing page.

$$H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}$$

Figure 3.7: Representation of a perspective transformation matrix

$$dst(x, y) = src\left(\frac{h_{11}x + h_{12}y + h_{13}}{h_{31}x + h_{32}y + h_{33}}, \frac{h_{21}x + h_{22}y + h_{23}}{h_{31}x + h_{32}y + h_{33}}\right)$$

Figure 3.8: A transformation from one plane to another plane

The ZXY coordinate system can be seen as a birds eye view of the field, while the camera captures another perspective of the same field, and the translations between these perspective planes can be expressed mathematically by a translation matrix. Figure 3.9 shows a synthetic generated birds eye view of the soccer field at Alfheim stadium with correct proportions. Now we will see how we can find the homography between this plane, the ZXY plane, and the planes captured by the cameras.

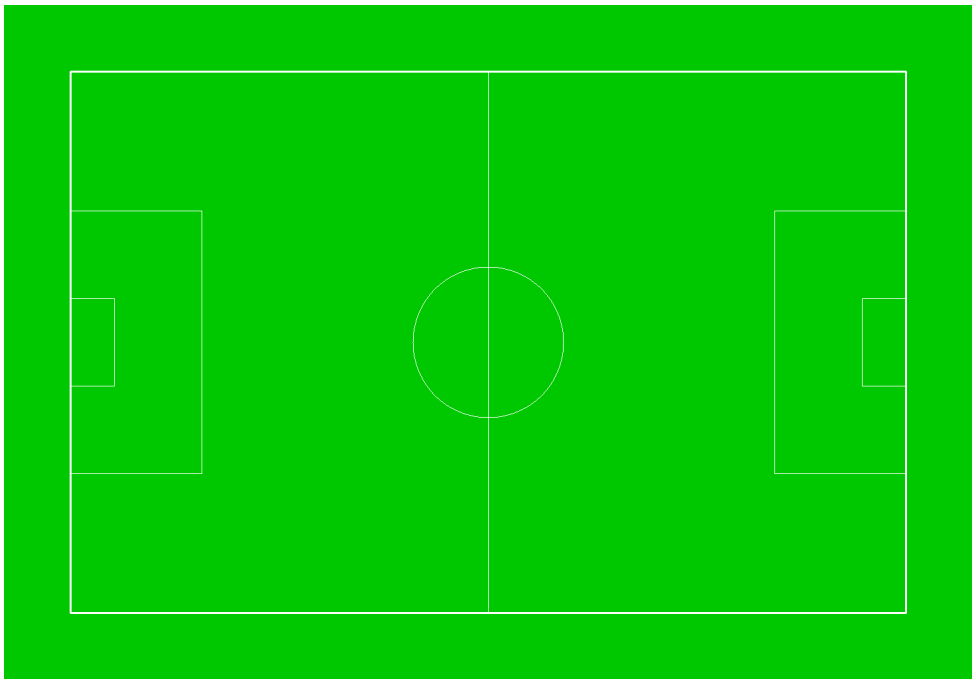

Figure 3.9: A synthetically generated view of Alfheim stadium, using correct proportions. Since ZXY uses these same proportions, it can also be seen as a generated view of the ZXY coordinate plane.

### Finding translation matrix

As mentioned in chapter 2, OpenCV has many methods and functions related to geometrical transformations. In particular, it has functionality to find the homography between two planes.

It is a function which takes a set of corresponding coordinates in each plane, in order to calculate the perspective transformation between them.

In our case we know the ZXY coordinates of many know positions, such as where the middle line crosses the sidelines, the corners etc. So the process of finding the transformation matrix involves picking pixel points in the camera images where the ZXY coordinates are known. Figure 3.10 shows an illustration of corresponding coordinates in the ZXY plane and the image plane.

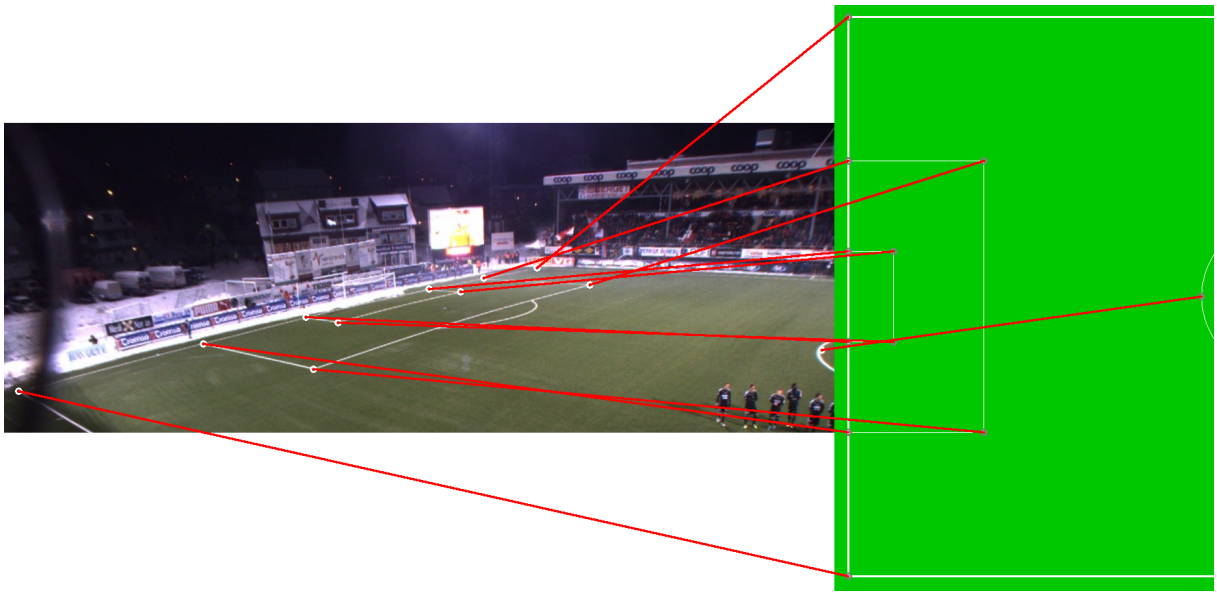

Figure 3.10: Corresponding coordinates in the ZXY plane and the image plane.

When the homography between two planes are found, each plane can be warped to fit the other. Figure 3.11 on the next page shows the 2nd camera view, warped and superimposed onto the ZXY plane and figure 3.12 shows the inverse.

### 3.3.2 Smart Camera selection

While knowing the exact pixel coordinates of a player is interesting, knowing which camera(s) a player is in, is of equal importance. If a player is sprinting across the field and scores a goal, we want to be able to generate a video sequence of that event which automatically selects the correct camera, i.e. the camera the player is in. In this section, we will describe an automatic camera selector, based on tracked players.

In order to detect if a player is currently in a cameras view, we can use the same transformation matrices used to translate between the coordinate systems. The width and height of each video

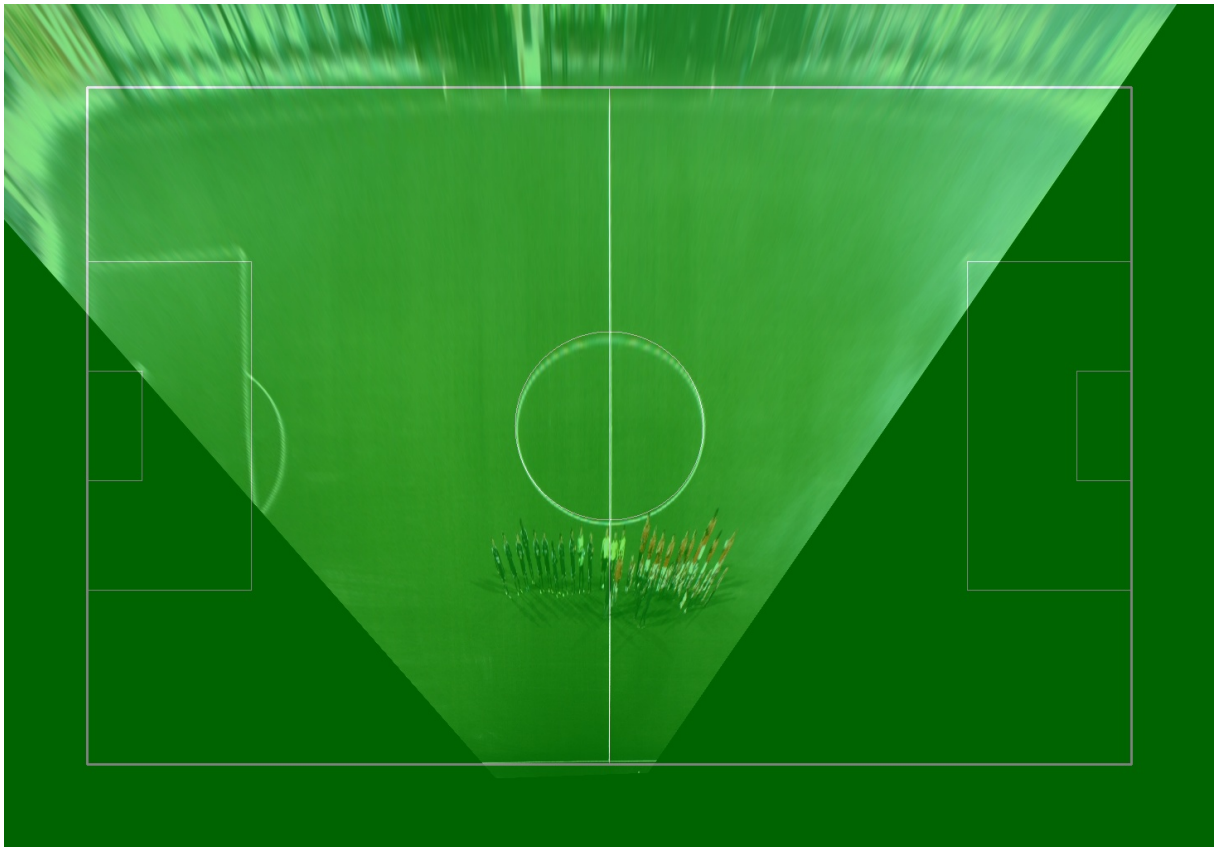

Figure 3.11: The image plane of the 2nd camera warped and superimposed on the generated ZXY plane.

frame is known and each camera has its own transformation matrix. To check whether a player is in the camera, one can calculate the pixel coordinates for that player using the transformation matrix for a specific camera. If you get pixel coordinates outside the width or height of the video frame, the player is located outside it. An implementation that finds the optimal camera for a single player is given in listing 3.22 on page 48.

When tracking multiple players we use the same routine and count the number of tracked players present in each camera and select the camera with the most tracked players. The implementation can be seen in figure 3.23 on page 49. This enables us to track a group of players, for example the defensive players. This enables us to create video sequences of for example an attack, by tracking the attacking players. Figure 3.13 on page 41 shows how the soccer positions and the color codes shows how players can be grouped together into defenders, forwarders and midfielders.

Figure 3.12: The ZXY plane, warped and superimposed onto the plane of the 2nd camera.

### 3.3.3 Implementation and performance

An implementation of the formula given in 3.7 on page 37 is shown in figure 3.15 on page 42. As we can see, transforming ZXY coordinates to pixel coordinates consist of nine additions, six multiplcations and two divisions which is hardly of any significance performance wise, on any modern machine. To make the performance test a bit more interesting, we have done the performance test tracking all 22 players in all four cameras at the same time. The computations can bee seen in table 3.14 on the next page. As we can see, the coordinate translation is hardly noticeable compared to the other components in the system.

## 3.4 Digital zooming

Digital zooming is a method used to decrease the apparent angel of view of an digital image or digital video. In contrast to optical zooming, there are no mechanical or optical components

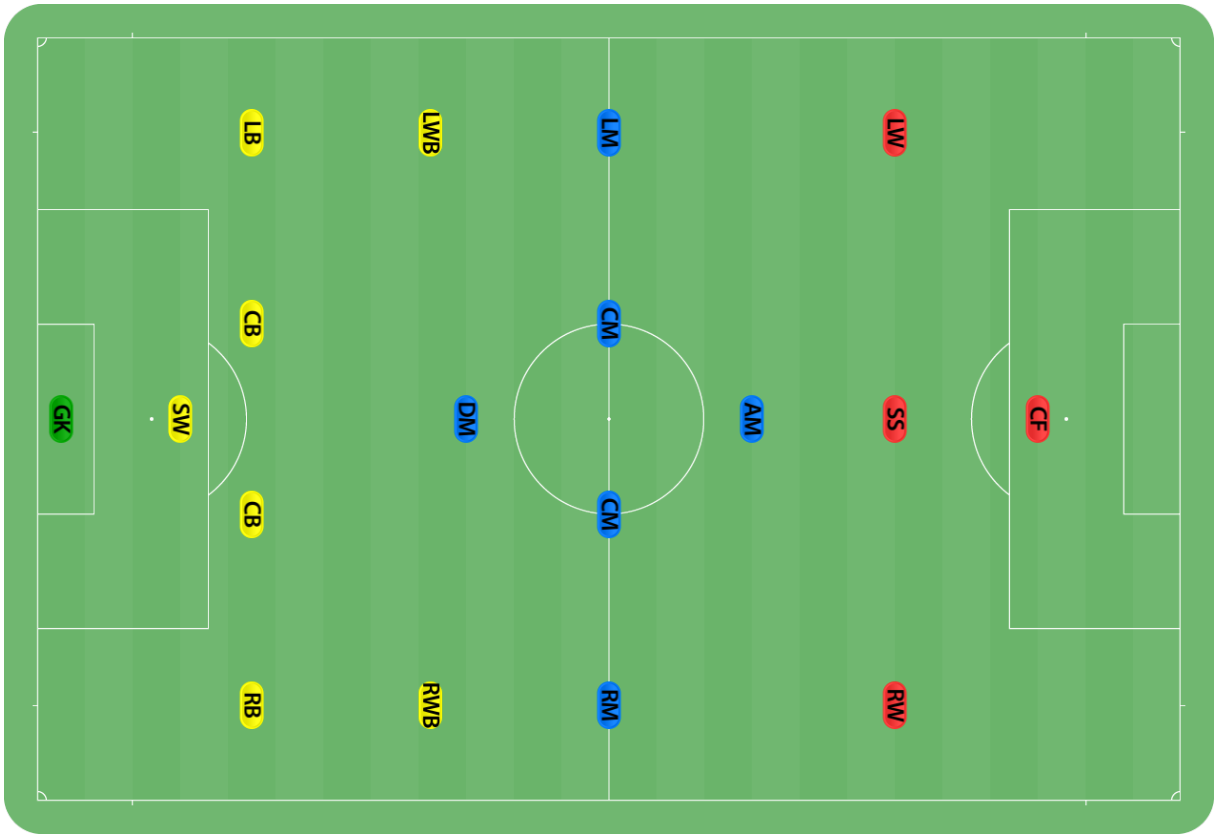


Figure 3.13: Modern player positions in soccer. Yellow is defender positions, blue is midfield positions and red are forward positions.

| | Min | Max | Mean |
|---|---|---|---|
| Coordinate translation | 0.007197 ms | 0.007776 ms | 0.007347 ms |

Figure 3.14: Function that finds the optimal camera for multiple players

involved. Digital zooming is accomplished by cropping an image to the desired area, often keeping the same aspect ration of the original image and then enlarging the image back to its original size by using some form of interpolation algorithm.

Since the first step is cropping the image, you loose a significant amount of pixels depending on the amount of zoom and the image's quality degrades. In addition to loosing pixels, the interpolation step might create jagged edges, depending on the interpolation type.

There are different form of interpolation techniques which gives a trade off between quality and performance. Figure 3.16 on page 43 shows a comparison between nearest neighbor interpolation (i.e no interpolation) and bi-cubic interpolation. As we can see, nearest neighbor shows severe pixelation effects compared to bi-cubic.

As presented before, we have the ability to track players and this can be utilized in combination

```
1  cv::Point2d getPoint2d ( int cam, double x, double y ) {
2
3    double tmpX, tmpY, tmpZ;
4    cv::Mat M = transformationMatrices[cam];
5    tmpX =  (M.at<double>(0,0)*x) + (M.at<double>(0,1)*y) + M.at<double
         >(0,2);
6    tmpY =  (M.at<double>(1,0)*x) + (M.at<double>(1,1)*y) + M.at<double
         >(1,2);
7    tmpZ =  (M.at<double>(2,0)*x) + (M.at<double>(2,1)*y) + M.at<double
         >(2,2);
8
9    return cv::Point2d(tmpX/tmpZ, tmpY/tmpZ);
10 }
```

Figure 3.15: Function that finds the optimal camera for a player

with digital zooming to create video sequences that follow specific players. By using the tracked player's pixel coordinates as center point for the cropped image, we can create a video sequence that keeps a player in the middle of the video at all times. Figure 3.17 on page 44 shows a tracked player in a video frame and the consecutive frame where the player is digitally zoomed in.

## 3.5 Evaluation and discussion

In this chapter, we have introduced a method of tracking players in video using a sensor network. We have showed that this solution is fairly lightweight in terms of computation power. However, the precision of the tracking depends on the multiple factors.

According to themselves, ZXY's tracking system installed at Alfheim stadium has a maximum accuracy error of one meter. This means that if a player is at position $ZXY(52.5, 34)$ (center field), both the X-axis and the Y-axis could be +- one meter. To illustrate this error in practive, figure 3.18 on page 45 shows a green square drawn using correct coordinates for the center field, while the red squares indicate an accuracy of one meter on both axis.

A more important part in terms of tracking accuracy error is the setup part where the transformation matrix is computed. The pixel coordinates are picked out manually which easily introduces human errors, and this is especially true for corners that are furthest away from the camera, where it is hard to pinpoint the exact position of the corner, due to the low resolution.

Another source of error for calculating the perspective transformation matrix is the cameras and the optics. In section 2.3.2 on page 22 we gave an introduction to camera calibration and the

Figure 3.16: Comparison of Bi-cubic (right image) and nearest neighbor interpolation (left image).

importance of correctly calibrated cameras and in section 2.4 on page 25 we showed that our cameras in our setup were not perfectly calibrated. Since our cameras does not have perfectly calibrated pictures, we can not produce perfect projective transformations between the ZXY coordinate system and the video cameras. Our initial tests in producing the transformation matrix were unsuccessful for the two of the cameras and an example of this can be seen in figure 3.19.

In order to get a correctly calculated homography, we did some trial and error attempts by removing and introducing new coordinate pairs for the calculation of the homography. The final result for all cameras can be seen in figure 3.21 on page 47 and it shows that our coordinate translation is quite accurate, although not perfect. When tracking players in the video, the tracking box does very rarely fail to capture the tracked player, however, it happens in some rare cases. Figure 3.20 on page 46 shows one of these cases. While there is an accuracy error in this certain case, tracking is successful in the same region in other cases. This shows that there are in fact inconsistencies in ZXY's positioning system, since calibration is constant.

One of the weaknesses of our approach is that the calibration is done manually and must be redone every time a camera is slightly moved. This could be needed for maintenance reasons, or the football could be kicked into one the cameras – not an unlikely scenario.

When it comes to following a player by switching cameras, these errors have little effect since they are relatively small compared to the frame size. However, when it comes to digital zooming

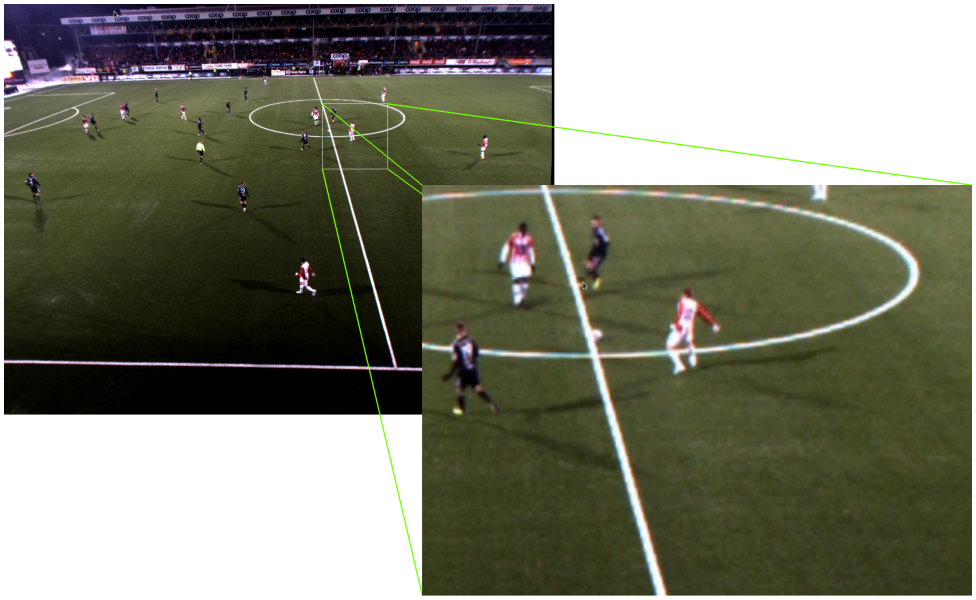

Figure 3.17: Digtal zooming using the tracked players pixel coordinate as center.

and following a player, it matters more. If the accuracy error is to extensive, the zoomed video will fail to capture the tracked player.

Another aspect with digital zooming is that it is relatively computational expensive depending on the type of interpolation done. Since our video is recorded at 30 frames per second, zooming must be done within $\frac{1}{30}$ second in order to keep the playback real-time. Figure 2.1 on page 22 shows a speed comparison between the different interpolation algorithms available in OpenCV. As we can see, only nearest neighbor- and bilinear-interpolation is fast enough in order to do digital zooming in real time.

## 3.6 Summary

In this chapter we have introduced a method for high precision tracking and automatic camera selection in multiple synchronized cameras. We have identified the importance of proper camera calibration in order to calculate an accurate homography between two planes, in order to get accurate player tracking. We have also showed digital zooming, which in combination of high precision tracking can be used to follow a player, and keep him centered in a video frame.

In the next chapter we will see how we can use video stitching in order to create a seamless stream that follows a player, instead of doing camera switching.

Figure 3.18: Illustration of maximum error caused by the inaccuracy of the tracking sensors. Green square is created using coordinates for center fields while the red squares shows error of +- 1 meter.

Figure 3.19: Our first attempt to find the homography between the ZXY coordinate plane and the image plane for camera one. The ZXY plane has been warped and superimposed onto the image plane using a miscalculated homography.

Figure 3.20: This figure shows an example of when the tracking box fails to capture the tracked player.

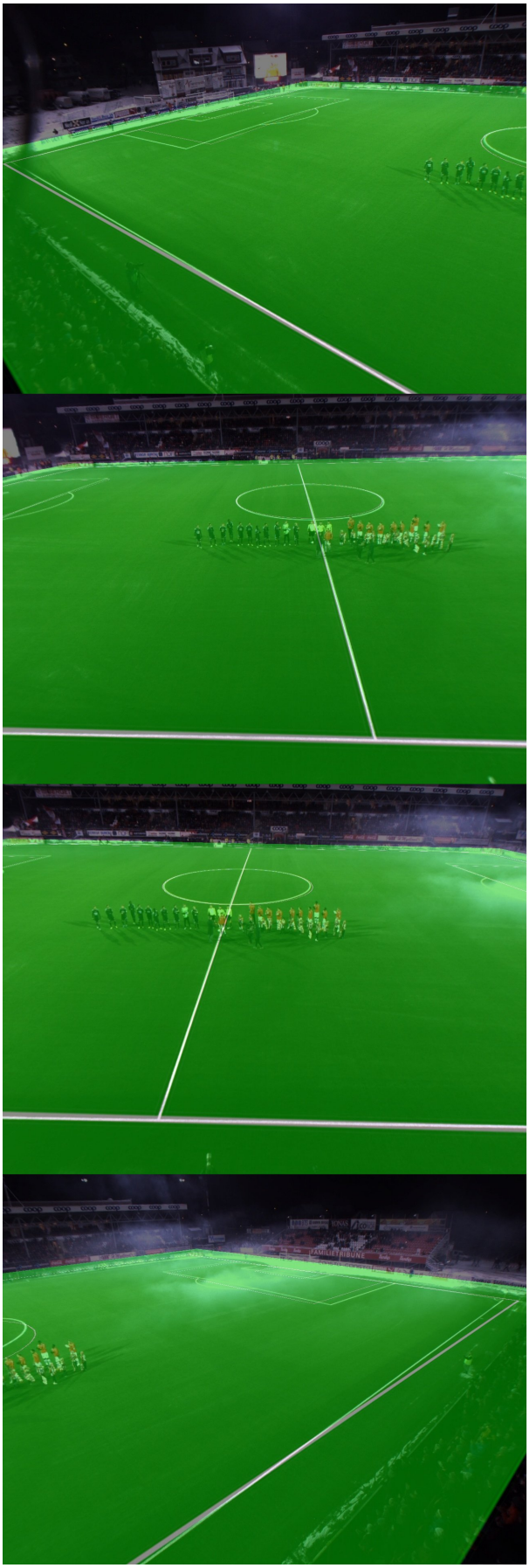

Figure 3.21: Final results, illustrating the accuracy of the coordinate translations.

```
int ZXYCoordinateHelper::getOptimalCamera ( double x, double y ) {
    int i;
    cv::Point2d tmp;

    if (x >52.5) {//We are on the right side of the field
        i=2;
    } else {//We are on the left side of the field
        i=0;
    }

    for (; i<nrCams; i++ ) {
        //Compute pixel coordinates from zxy coordinates
        tmp= getPoint2d (i,x,y) ;

        //Check whether we're outside the video frame
        if ( tmp.x < 0 || tmp.x >= STD_WIDTH ) {
            continue;
        }
        //Check whether we're outside the video frame
        if ( tmp.y < 0 || tmp.y >= STD_HEIGHT ) {
            continue;
        }
        return i;
    }
    return -1;
}
```

Figure 3.22: Function that finds the optimal camera for a single player

```
int ZXYCoordinateHelper::getOptimalCamera(vector< std::vector< double > >
    coordinates)
{
    int camCount[4];
    camCount[0] = 0;     camCount[1] = 0;
    camCount[2] = 0;     camCount[3] = 0;
    int i = 0;
    //Loop through the coordinates of tracked players
    for (unsigned int j=0; j< coordinates.size(); j++) {
        vector<double> tmp1 = coordinates[j];
        double x = tmp1[0];
        double y = tmp1[1];
        cv::Point2d tmp;

        for ( i=0; i<nrCams; i++ ) {
            tmp= getPoint2d (i,x,y);
            //Check whether we're outside the video frame
            if ( tmp.x < 0 || tmp.x >= STD_WIDTH ) {
                continue;
            }

            //Check whether we're outside the video frame
            if ( tmp.y < 0 || tmp.y >= STD_HEIGHT ) {
                continue;
            }
            camCount[i]++;
        }
    }

    int max = 0;
    int theCam = 0;

    //Find the camera which has most tracked players
    for (i=0; i<nrCams; i++) {
        if (camCount[i]> max) {
            max = camCount[i];
            theCam = i;
        }
    }

    return theCam;
}
```

Figure 3.23: Function that finds the optimal camera for multiple players

# Chapter 4

# Video Stitching

In the previous chapter we introduced a method for high precision video tracking using a sensor network. One of the effects with following a player in multiple cameras is that you have to switch cameras. In our setup we have used only four cameras, which is barely enough to cover the entire soccer field and one can image a system that uses many more cameras. If you are following a player which is running over the entire field, this could involve a lot of cameras switching which could be confusing for a viewer. This is why we wanted to look at how we can stitch together frames from multiple cameras in order to create a seamless video where you wouldn't have to switch cameras at all when following a player.

In this chapter, we will introduce a method for stitching video from static cameras where their orientation does not change in relation to each other. This simplifies our problem since we only need to do *image registration* (see section 4.1) for the first frame in a video stream. Video is just a series of images and when introducing video stitching, we will introduce it under the term – image stitching. Image stitching is the process of combining multiple photographic images with overlapping fields of view, to produce a segmented panorama or high-resolution image.

## 4.1 Basic stitching theory

As mentioned, image stitching is the process of combining multiple photographic images into a *segmented panorama* or *high-resolution* image. In this thesis we will only look at how we can combine images into a segmented panorama. The image stitching process can be divided into three main components - *image registration*, *calibration* and *blending*.

### 4.1.1 Image stitching

Image registration involves matching features in a set of images or using direct alignment methods to search for image alignments that minimize the sum of absolute differences between overlapping pixels. When using direct alignment methods one might first calibrate ones images to get better results. Additionally, users may input a rough model of the panorama to help the feature matching stage, so that - for example - only neighboring images are searched for matching features. Since there are smaller group of features for matching, the result of the search is more accurate and execution of the comparison is faster.

Image calibration aims to minimize differences between an ideal lens models and the camera-lens combination that was used, optical defects such as distortions, exposure differences between images, vignetting, camera response and chromatic aberrations. If feature detection methods were used to register images and absolute positions of the features were recorded and saved, stitching software may use the data for geometric optimization of the images in addition to placing the images on the panosphere.

Image blending involves executing the adjustments figured out in the calibration stage, combined with remapping of the images to an output projection. Colors are adjusted between images to compensate for exposure differences. If applicable, high dynamic range merging is done along with motion compensation and deghosting. Images are blended together and seam line adjustment is done to minimize the visibility of seams between images.

### 4.1.2 Map projections

For image segments that have been taken from the same point in space, stitched images can be arranged using various *map projections*. There exists many methods of map projections, and a map projection is essentially a method of representing the surface of a sphere or any other three-dimensional body, on a plane. There are essentially three categories of map projections.

**Rectilinear projection**

Rectilinear projection is where the stitched image is viewed on a two-dimensional plane intersecting the panosphere. Lines that are straight in reality are *shown as straight lines* regardless of their direction. As a result of this, wide views of $120^o$ and above suffers from severe distortions near the edges of the stitched image [24].

**Cylindrical projection**

Cylindrical projection is where the stitched image shows a $360^o$ field of view, and a lim-

ited vertical field of view. The idea behind cylindrical projection is that the image should be viewed as though the image is wrapped on the inside of a cylinder. This affects horizontal lines when they are viewed on a two dimensional plane, and they appear curved while vertical lines remain straight. The wrapped image can be displayed with special software to create a view that does not have distortion, or on a two-dimensional plane with distortion [24].

**Spherical projection or equirectangular projection**

Spherical projection is similar to cylindrical projection, where the stitched image shows a $360^o$ horizontal and $180^o$ vertical field of view. Panoramas in this projection is meant to be view as though you where standing inside an sphere and that the stitched image is projected on the inside of the sphere. When viewed on a two dimensional plane, horizontal lines appear curved while vertical lines remain vertical. Google maps is an example of this kind of projection where they use a cylindrical projection called *mercator projection*.

### 4.1.3 Challenges of image stitching

One of the main challenges of image stitching is parallax errors. Parallax errors is an effect that arises when you are capturing images of the same scene from different viewpoints. Figure 4.1 on the following page illustrates this effect. Usually, the way to fix parallax errors when making image panoramas, is to keep the camera stationary and only moving it along the axis of the entrance pupil of the camera. Since we are using multiple cameras for our stitching, this is not possible and as we will see, our stitching will suffer from these parallax errors.

Another challenge is when you capture images or video at different points in time and try to stitch them. As already described, we have a trigger box which synchronizes the camera shutters and all our frames are captured at exactly the same time.

In this chapter we will look into an automatic stitcher involving all the steps above and we will introduce our own stitcher which has a more manual approach.

## 4.2 OpenCV's autostitcher

As already mentioned, OpenCV is an open source computer vision library. OpenCV offers a stitching module which is based on Brown's [26] automatic panoramic image stitcher. The full pipeline of the stitcher can be seen in figure 4.2 on page 55. Each component in the pipeline can

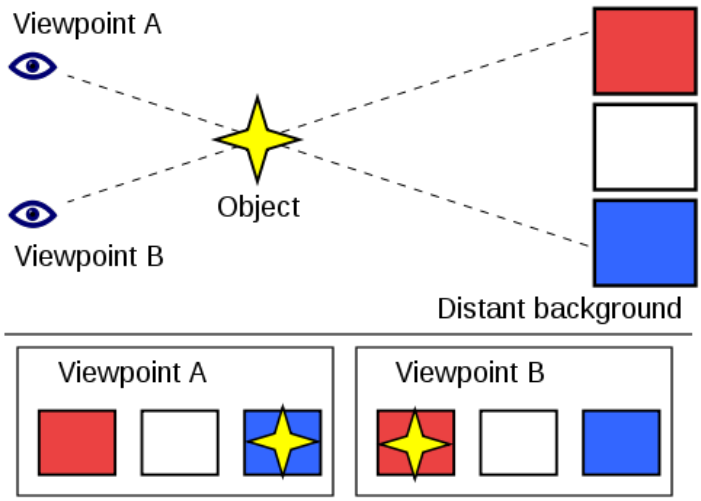


Figure 4.1: A simplified illustration of the parallax of an object against a distant background due to a perspective shift. When viewed from "Viewpoint A", the object appears to be in front of the blue square. When the viewpoint is changed to "Viewpoint B", the object appears to have moved in front of the red square [25].

be fine tuned with parameters in order to create a better output result. We will not be going into all the details of OpenCV's autostitcher, but rather demonstrate how it can be used as a tool, and also to demonstrate various map projections. At the time of writing this thesis, the stitching module in OpenCV was under development, and it did have bugs and non deterministic behavior on the same data input. Figure 4.4 on page 56 shows a resulting stitched image using cylindrical projection while figure 4.5 on page 57 shows another result using spherical projection.

## 4.3 Homography stitching

In section 3.3 we explained how we can use projective geometry to translate ZXY's coordinate system into pixel coordinates and we also described that two views of the same plane are related by a homography. In this section, we will investigate how a homography can be utilized in order to stitch images together. The process is the same as explain in chapter 3, however we will briefly summarize it here. The first step is to find corresponding pixel points in order to compute the homography between the two planes. When the homography is calculated, the image can be warped in order to fit the plane of the second image. However, if we use this matrix directly, we produce an image that only contains the pixels that exist in the camera we are converting to. Figure 4.6 shows the first camera view and 4.7 on page 58 shows the second camera. Figure 4.8 shows the first camera warped to fit the second view. As we can see, most

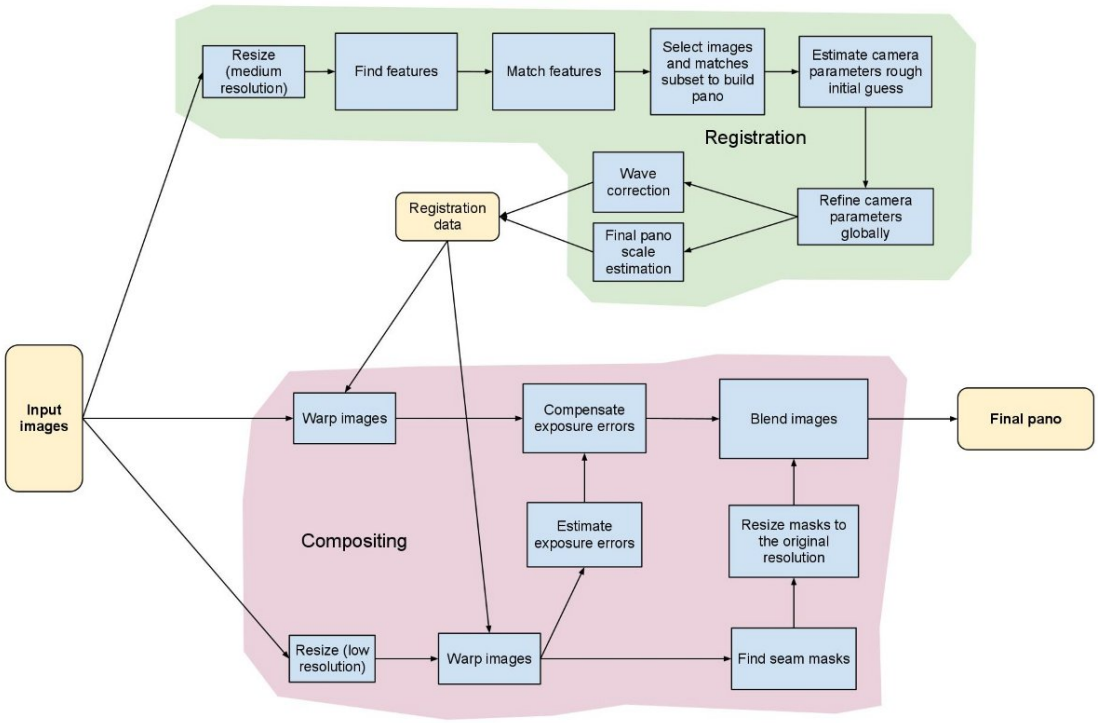


Figure 4.2: This figure illustrates the stitching module pipeline implemented in the OpenCV library. The implementation is based on Brown and Lowe's autostitcher [26].

of the image is lost.

In order to get the full view when warping the image, we need to calculate the new size of the full image. This is done by taking the leftmost pixel coordinates, namely $pix(0,0)$ and $pix(0, imageHeight)$ and translate them with the transformation matrix. This gives us the pixel coordinates in the image plane we are transforming to. If the coordinates are outside the image, it means that the image must be padded in order to include all the pixels in the warped image. If we take the minimum X coordinate from this calculation and then expand the image with this size, we will get the full view. Figure 4.9 shows this done to the first camera.

In order to stitch the two cameras together, the second camera has to be padded with the same amount of the first camera so that their sizes match. When this is done, the images can be superimposed on each other in order to create a stitched version. Figure 4.11 shows this. Figure 4.10 shows each of the 4 cameras, warped and padded and it also shows the four warped and padded images with overlapping regions.

Figure 4.3: An image panorama create with OpenCV's autostitcher, using planar projection

Figure 4.4: An image panorama create with OpenCV's autostitcher, using cylindrical projection

### 4.3.1 Algorithm

- Image registration
  - Pick a camera that acts as a head camera.
  - Find corresponding points between each of the other cameras and the head camera.
  - Compute the projective transformation matrix (homography) between each camera and the head camera.
  - Using the transformation matrices, compute the padding for each camera.
  - Choose where the seams of the stitched image should be drawn. The seams should be drawn in the overlapping regions.
- Image stitching
  - Warp each camera using their calculated transformation matrix.

Figure 4.5: An image panorama create with OpenCV's autostitcher, using spherical projection

- Pad each image left and right so that it is the same size as the final view.
- Create the final panoramic view by choosing pixels from the correct camera according to where the seams are drawn.

Figure 4.6: A view captured by the first camera. Note that the image is debarreled.

### 4.3.2 Automatic setup

As we saw in the previous section, the image registration step must be done manually by picking out corresponding points in two images. An alternative approach to this is using *feature matching* in order to find corresponding pixel points. *Feature detection* refers to methods that aim at

Figure 4.7: A view captured by the second camera. Note that the image is debarreled.

computing abstractions of image information and making local decisions at every image point whether there is an image feature of a given type at that point or not. While feature matching aims to find corresponding features in a set of images.

*Feature matching* is a well researched topic and we will not be going into the details of feature matching techniques and algorithms. We will however show how a feature detection algorithm known as SURF (Speeded Up Robust Feature) [27] can be used to find corresponding points in order to compute the projective transformation matrix between two planes.

SURF was first presented by Herbert et al. [27] in 2008. SURF is local feature detector used to detect interesting areas in an image like edges, corners etc. After the features are found in each image, we need to match the features to find corresponding matches and this can be done be done by any nearest neighbor search.

OpenCV has an implementation of SURF as well as an interface to FLANN (Fast Library for Approximate Nearest Neighbors). Figure 4.12 shows corresponding matches after running FLANN on the features found by SURF. As we can see there are many false matches and this is solved when we compute the homography (projective transformation matrix). When we are computing the homography, we have the option to turn on RANSAC (RANdom SAmple Consensus).

RANSAC is an iterative method to estimate parameters of a mathematical model from a set of observed data which contains outliers. A simple example is fitting of a line in two dimensions to a set of observations. Assuming that this set contains both inliers, i.e., points which approximately can be fitted to a line, and outliers, points which cannot be fitted to this line, a simple least squares method for line fitting will in general produce a line with a bad fit to the inliers.

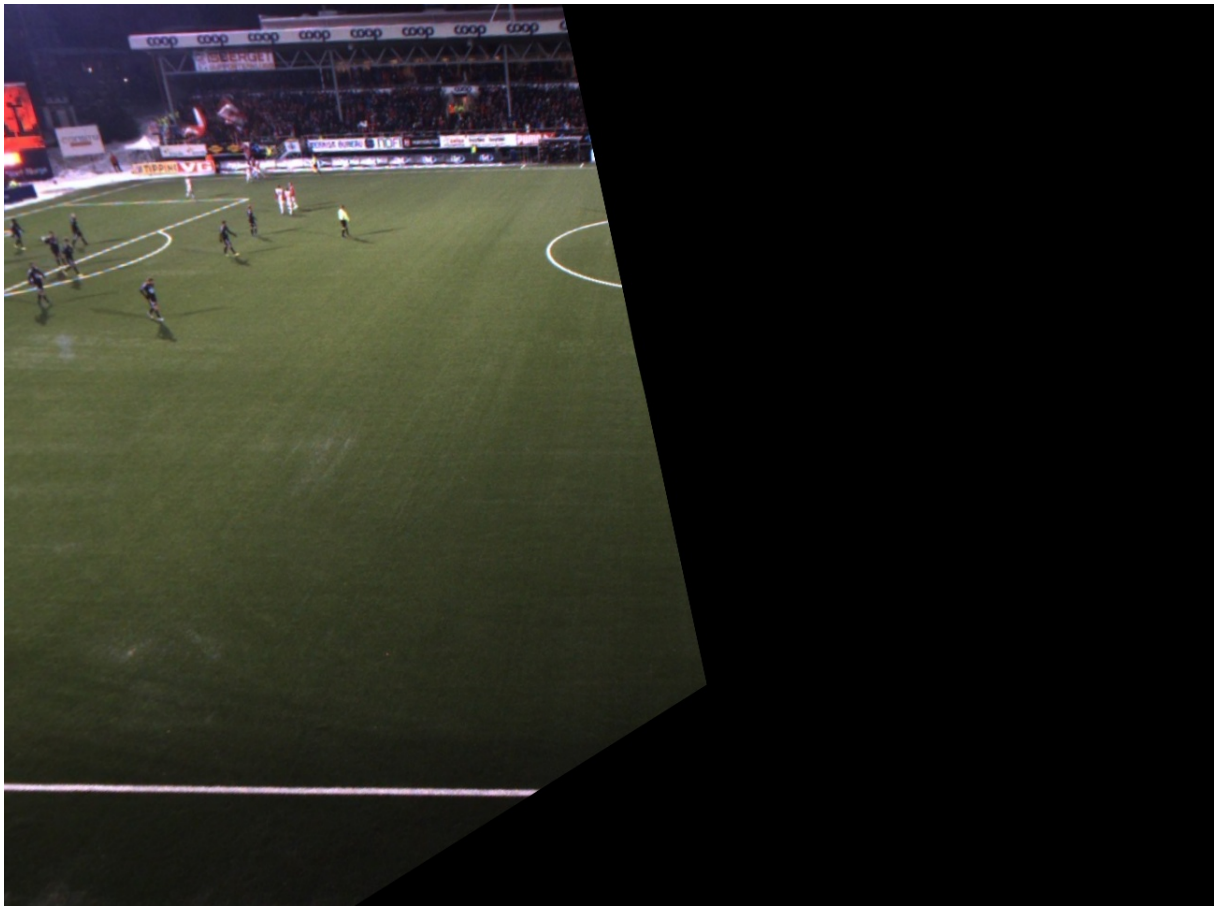

Figure 4.8: The first camera is warped to fit the plane of the second camera. However, padding is not calculated, leaving only pixels that are also captured from the second camera.

Figure 4.13 on page 64 shows an illustration of this example and how outliers have no influence on the result. The final result of a homography found by using this described methods can be seen in figure 4.8. The matching points shown that were use to compute the homography is shown in figure 4.12 on page 64.

## 4.4 Tracking in stitched video

The problem with stitching with OpenCV is that we have no control of how the stitched image is projected and the coordinates in this plane. Also, the result varies from data-set to data-set because of the feature matching is used to align the methods. Since feature matching is dependent on the contents of the image, the output result varies. OpenCV's stitching module is made to work as an automatic panorama stitcher and not for stitching a sequence of images the same way. Because of this, we have not tried to track players in the output from OpenCV's

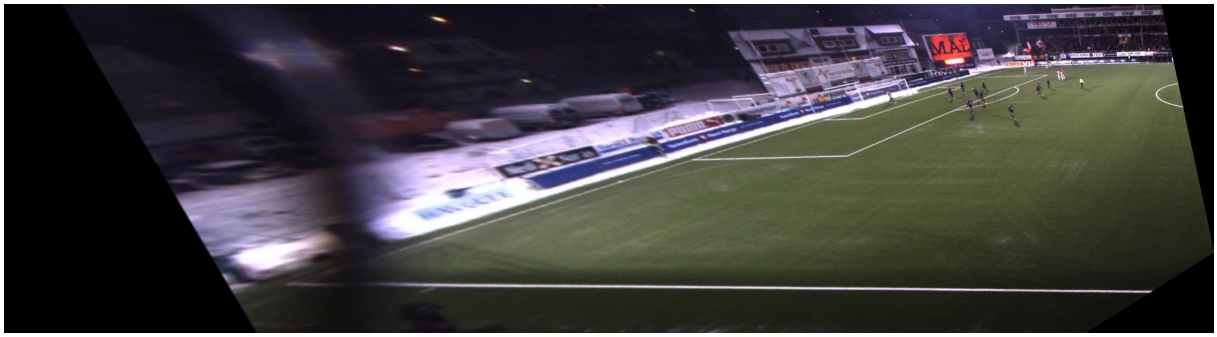

Figure 4.9: The first camera is warped to fit the plane of the second camera with padding calculated. Note that the image is cropped for display purposes.

stitcher.

However, in our own stitching, we have full control of how the images are projected and we can therefore do high precision easier. The approach here is the same as in chapter 3, where we pick known points in the image plane and corresponding ZXY coordinates and compute the homography between the two planes. However, since the stitched is composed of several warped planes and they are not perfectly aligned, it is hard to find a perfect projective transformation between the ZXY coordinate plane and the stitched plane. Figure 4.14 on page 65 shows the ZXY plane superimposed onto the stitched image. As we can see, there are accuracy errors.

## 4.5 Discussion and evaluation

### 4.5.1 OpenCV's autostitcher

As our first attempt to stitch videos and to create a reference we wanted to test the results of OpenCV's auto stitcher. OpenCV's autostitcher was made with images in mind, and not for stitching video and its performance spans from a couple of seconds to minutes, depending on parameters for every frame. In this thesis, we have used OpenCV's stitcher in order to demonstrate how different results and map projections and also, how features such as adjusting exposure between images and image blending (to hide stitching seam) improve the visual quality of the result. In our opinion, OpenCV's autostitcher creates more visually pleasing results compared to our homography stitcher, which only aligns the image geometrically correctly.

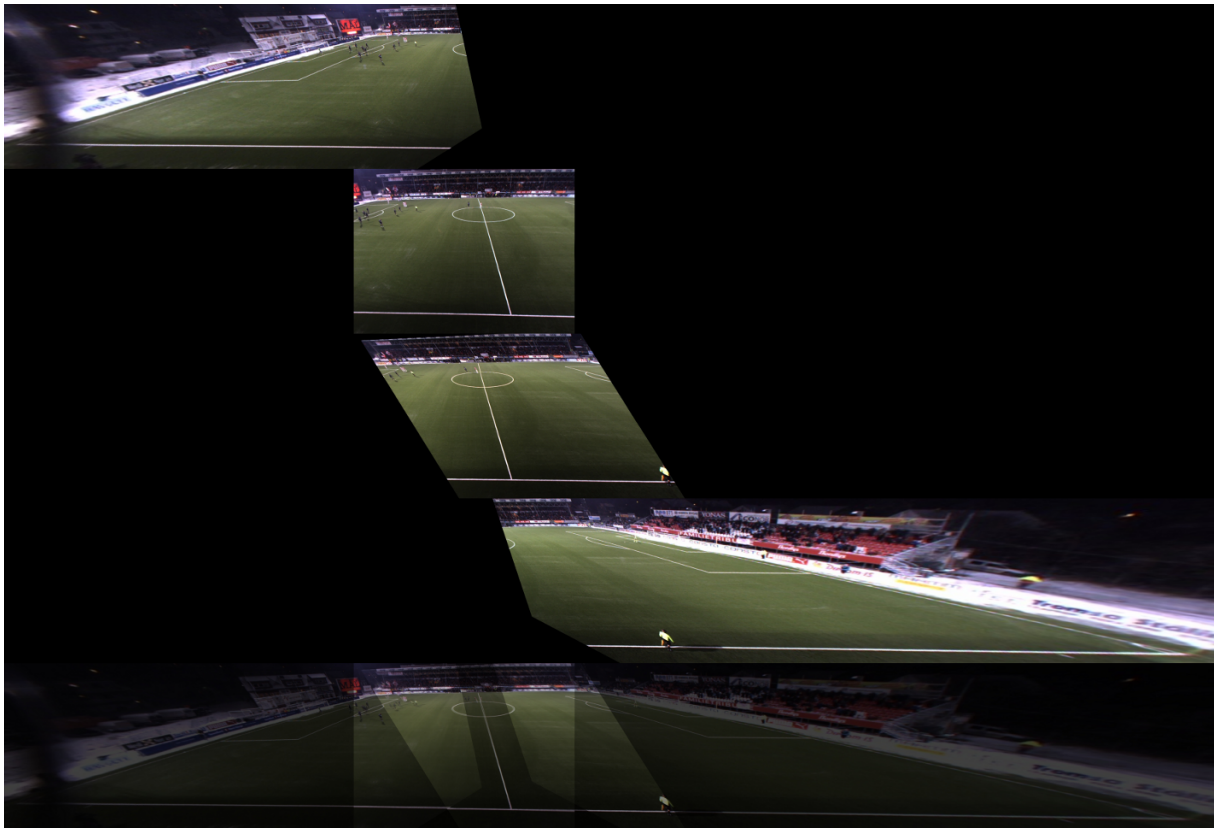

Figure 4.10: This figure shows each of the 4 cameras, warped and padded to fit the view of the second camera. Lastly, it shows these 4 view superimposed on each other. The highlighted areas show where the views overlap.

### 4.5.2 Homography stitcher

Our second approach, we utilized the property that any two images of the same planar surface is related by a homography. However, this relation assumes the pinhole camera model where no image distortion because of lenses. In section 2.3.2, we showed that a camera can be calibrated to minimize lens distortion caused by imperfections in a lens, however we concluded that our calibration was imperfect. This makes finding a homography between planes difficult and error-prone, which affects the stitched result.

Another problem we have identified, are parallax errors. OpenCV's autostitcher has functionality for selecting seams at places where parallax errors are less obvious. However, when stitching video, players are bound the run through a seam and parallax errors will become prominent. Figure 4.15 show the stitching seam going through a player and figure 4.16 illustrates parallax effects that will occur.

### 4.5.3 Tracking

In the previous chapter, we introduced high precision tracking using a sensor network which we showed to be fairly accurate. In figure 4.14, we can clearly see that there are accuracy errors. However, the tracking boxes manages to capture the players, although unable to keep the players centered in the tracking box. An example of this is seen in figure 4.17.

## 4.6 Conclusions

In this chapter we have shown two implementations of video stitching – one based on an automatic panorama stitcher and one with a more manual image registration step, using the properties of the pinhole camera model and the homography relationship between planes.

In earlier chapters we have covered video capture and player tracking and in the next chapter we will see how we can integrate these three components, video capture, player tracking and video stitching, into the next generation soccer analysis system.

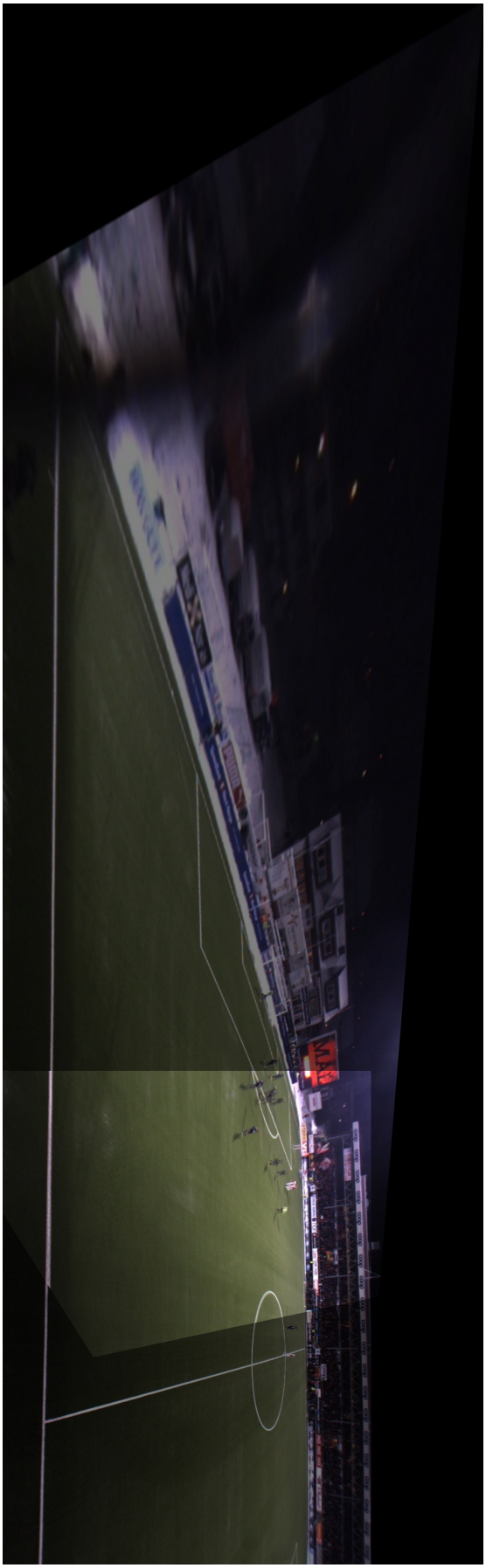

Figure 4.11: Shows figure 4.9 and 4.7 superimposed on each other. The highlighted area shows where they overlap.

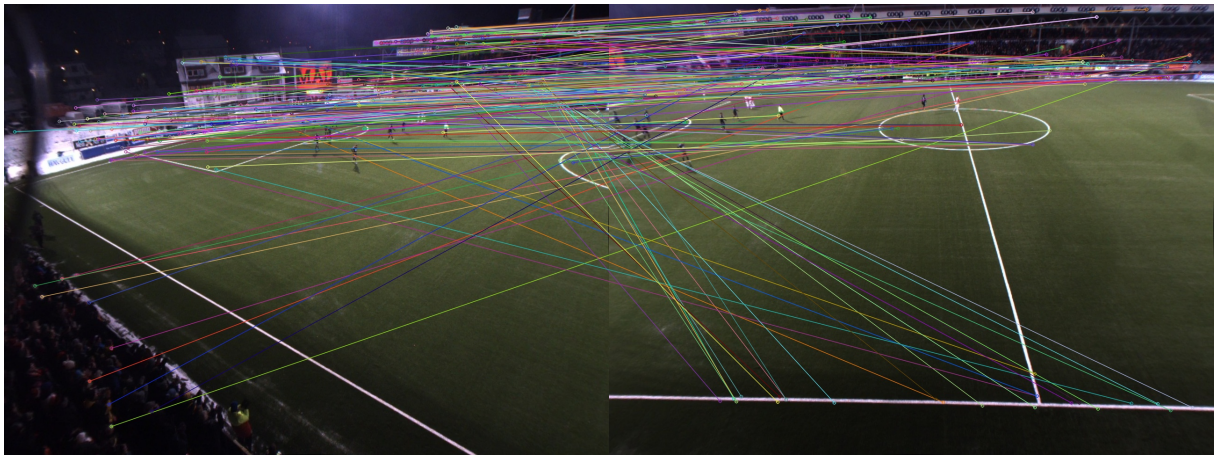

Figure 4.12: An example of feature points found using SURF and then matched by using FLANN.

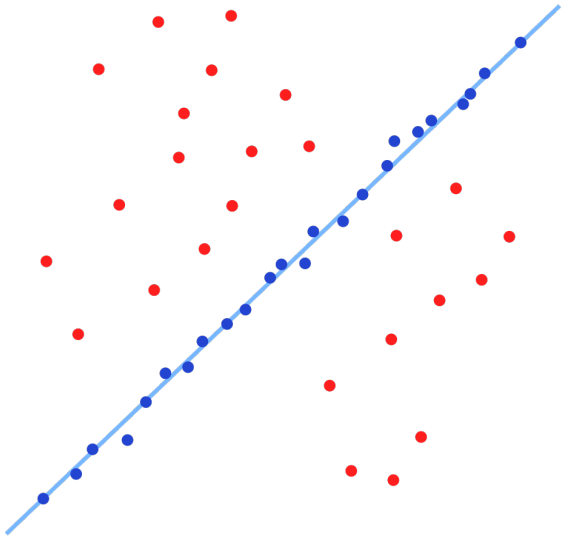

Figure 4.13: Fitted line with RANSAC, which shows how outliers have no influence on the result.

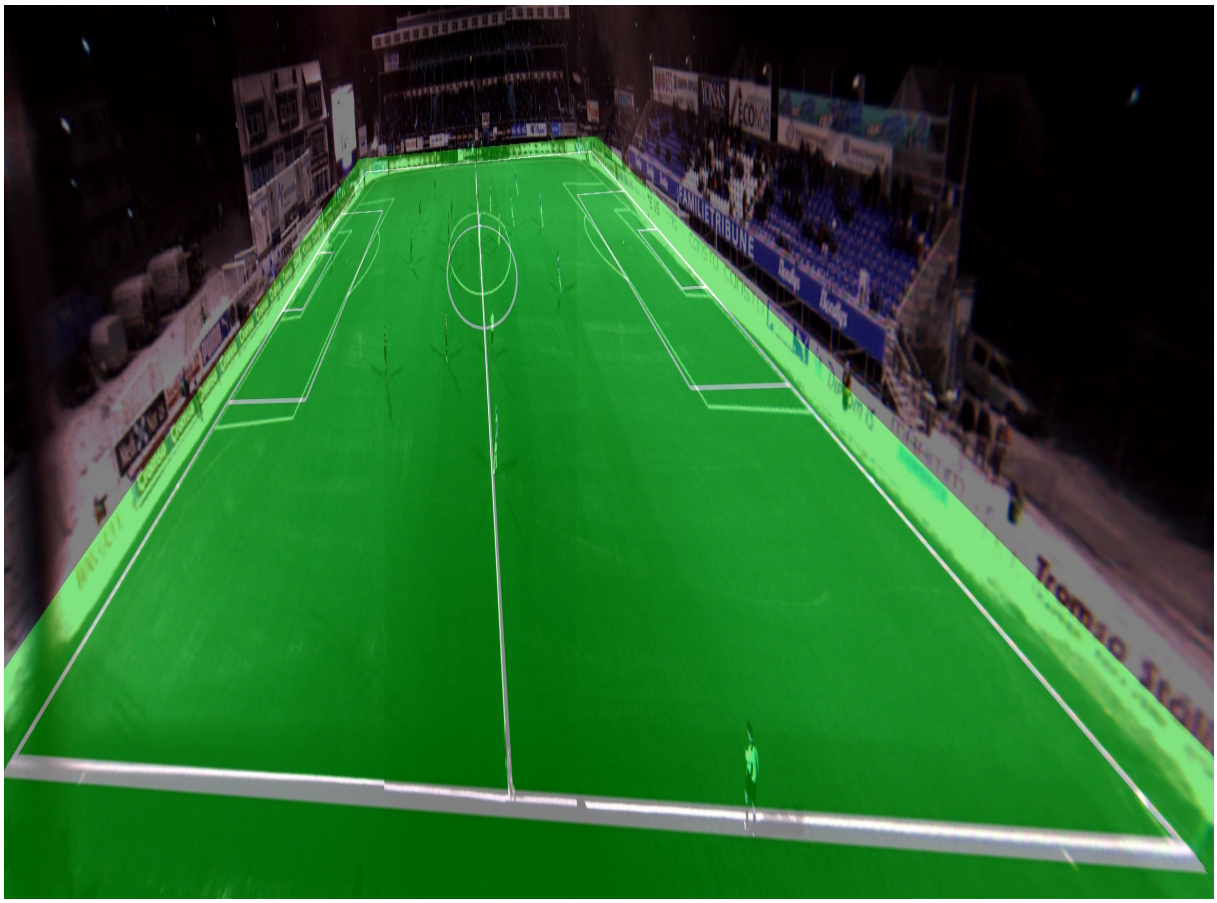

Figure 4.14: An image illustration the accuracy error for tracking in stitched video.


Figure 4.15: An image showing parallax errors where a player appear on both sides of the seam.

Figure 4.16: An area where three of the cameras overlap which illustrates parallax errors. The most highlighted area is composed of three images superimposed onto each other, the middle one consists of two, while the dark one consist of only one.

Figure 4.17: Example of player tracking in stitched video. The video is cropped for displaying purposes

# Chapter 5

# Video annotation and application

In this chapter, we will describe how the components described in the previous chapters, and an analytic sub-system, can be integrated into an soccer analysis system. First we will introduce our system, which has been prototyped, then we will look at some applications for such a system.

## 5.1 Next generation sports analysis

Bagadus is a prototype that is built in cooperation with the Tromsø IL soccer club together with the ZXY sports tracking company for soccer analysis. An overview of the architecture and how the different components interact is given in figure 5.1. The Bagadus system is divided into three different subsystems which are integrated in our soccer analysis application.

### 5.1.1 Sensor subsystem

Tracking people through camera arrays has been an active research topic for several years, and several solutions has been suggested. The accuracy of such systems are improving, but they are still giving errors. For stadium sports, an interesting approach is to use sensors on players to capture the exact position. ZXY Sports Tracking [5] provides such a solution where a sensor system submits position information at an accuracy of one meter at a frequency of 20 Hz. Based on these sensor data, statistics like total length ran, number of sprints of a given speed, etc. can be queried for, in addition, to the exact position of all players at all times.

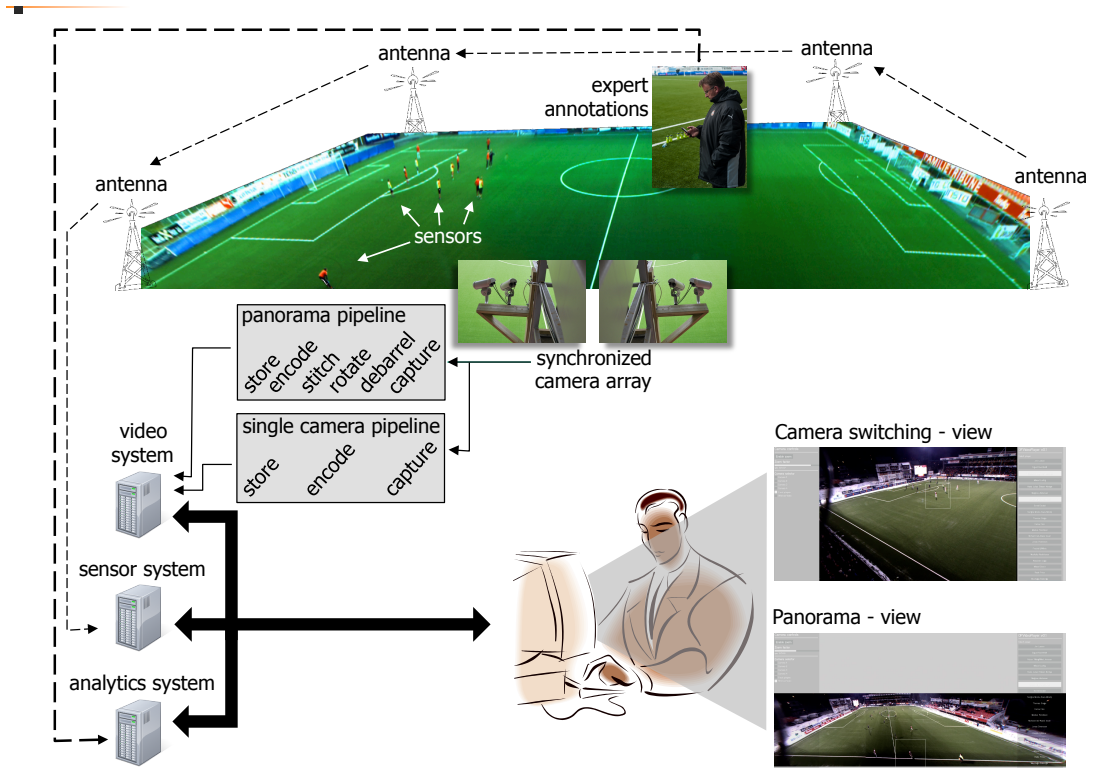


Figure 5.1: Architecture

### 5.1.2 Analytics subsystem

Coaches have for a long time analyzed games in order to improve their own team's game play and to understand their opponents. Traditionally, this have been done by making notes using pen and paper, either during the game or by watching hours of video, e.g., some clubs even hire one person per player. To reduce the manual labor, we have integrate a soccer analytics subsystem which equip a user with a device like a mobile phone or a tablet where for example buttons can fast register predefined events and/or events can be annotated textually. The registered event is then stored in an analytics database together with the time for later retrieval. Two different systems, Vuvuzela [28] and Muithu [29], have been tested and deployed at Alfheim stadium and have made a proof of concept integration with Vuvuzela.

### 5.1.3 Video subsystem

To record high resolution video of the entire soccer field, we have installed a camera array consisting of 4 Basler industry cameras with a 1/3-inch image sensor supporting 30 fps and a resolution of 1280×960. The cameras are synchronized by an external trigger signal in order to enable a video stitching process that produces a panorama video picture. The cameras are

mounted close to the middle line under the roof covering the spectator area. With a 3 mm lens, each camera covers a field-of-view of about 68 degrees, i.e., all four cover the full field with sufficient overlap to identify common features necessary for camera calibration and stitching.

The video subsystem supports two different playback modes. The first allows playing video from individual cameras where the view switches automatically between the cameras, i.e., manually selecting a camera or automatically following players. For this mode, the video data from each camera is stored and encoded separately. The second mode plays back a panorama video stitched from the 4 camera feeds. The cameras are calibrated in their fixed position, and the captured videos are each processed in a capture-debarrel-rotate-stitch-store pipeline. This means that due to the 3 mm fish-eye lens, we must correct the images for lens distortion in the outer parts of the frame. Then, since the cameras have different position and point at different areas of the field, the frames must be rotated and morphed into the panorama perspective. Finally, the overlapping areas between the frames are used to stitch the 4 videos into a $7000 \times 960$[1] panorama picture before storing it to disk. Currently, the panorama pipeline is non-real-time, but we are currently working on optimizing, parallelizing and offloading operations to multiple cores and GPUs.

### 5.1.4 System integration

The Bagadus application implements and merges many well-known components to support a particular application scenario. However, the combination of different technologies raises requirements of both spatial and temporal synchronization of multiple signals from different sources. The main novelty of our approach is therefore the combination and integration of components enabling automatic presentation of video events based on the sensor and analytics data which are synchronized with the video system. This gives a threefold contribution: 1) a method for spatially mapping the different coordinate systems of location (sensor) data and video images to allow for seamingless integration, 2) a method to record and synchronize the signals temporally to enable semantic search capabilities, and 3) the integration of the entire system into an interactive application that can be used online and offline.

Thus, Bagadus will for example be able to automatically present a video clip of all the situations where a given player runs faster than 5 meters per second or when all the defenders were located in the opponent's 18-yard box (penalty box). Furthermore, we can follow single players and groups of players in the video, and retrieve and play out the events annotated by expert

[1] Note that the panorama resolution is larger than what the 4 cameras deliver individually due to the warping, changing perspective, rotation and stretching of the original images.

users. Thus, where sport analytics earlier used a huge amount of time of manual labor, Bagadus is an integrated system where the required operations and the synchronization with video is automatically managed.

### 5.1.5 Demo application

In order to demonstrate the integration of such a system we have developed a demo application. We show how we have integrated a camera array, sensor data and professional soccer analytics' annotations into one application. We demonstrate the prototype system using an example data set recorded at Alfheim Stadium (Tromsø, Norway). The demo is a real-time application, where all the processing is done in real time and user can experiment with the system, follow and zoom in on particular player(s), and and play back events from the game in panorama video and camera switching mode. Figure 5.2 shows the user interface of our demo. The reader is encouraged to view the demo video of this application [2].


Figure 5.2: The user interface of our interactive demo application.

[2] http://home.ifi.uio.no/paalh/videos/bagadus.mov

### 5.1.6 Bagadus as a soccer analysis tool

With Bagadus, we have created a powerful tool for automatically generating video sequences of tagged events. As a proof of concept, we integrated our system with Vuvuzela in order to generate video sequences of tagged events. Using a simple web-sever, parsing XML-data event data sent by the Vuvuzela system, our system created video sequences of Vuvuzela-tagged events and uploaded them to a web server. One of the example events we used, was one of the goals scored by Sigurd Rushfeldt. By using Sigurd's positioning data and tracking him, we could automatically produce a video sequence of his goal.

In chapter 2, when introducing ZXY's sensor system, we mentioned that ZXY's system also records data like step frequency, heart rate and speed. This allows coaches to monitor a players performance over time. Such a system, combined with video is even more powerful.

One could imagine a scenario where a player gets a knee injury, and has to stop playing for a while. After some months of recovery, he plays a match. According to his performance profile, his performance has deteriorated and he is not sprinting as fast as he was before his injury. With Bagadus, a coach has the tool generate video sequences of all sprints made by that player, to further investigate and evaluate if said player should be allowed to play, or stay in recovery.

Video feedback is a very powerful tool for a coach, because it allows a coach to *show* an athlete what he doing wrong, rather than just telling him.

## 5.2 Bagadus for home entertainment

Currently, there are many services delivering video over the Internet, ranging from live event streaming replacing the traditional television broadcasting, to uploading, sharing and delivery of personalized content. Many people therefore think the time is ripe for radical changes in entertainment and news production systems in order to integrate emerging Internet technologies and services frequently used by the new generation of multimedia consumers.

In this section, we will discuss how Bagadus can be used meet the requirements of users familiar with services like Facebook, YouTube, blogs, etc. and to improve search and personalization functionality. For instance, YouTube is overloaded with fan videos like "All the Free Kicks from David Beckham" and "Lionel Messi - Top 10 goal". These video are manually edited together, but in this section we will discuss how Bagadus can be used to search of "All penalty kicks from Sigurd Rushfeldt".

### 5.2.1 Video search

A quickly emerging service is sports streaming, both live and on-demand, where users may retrieve whole events or predefined highlights, and where the video data is presented with all kind of statistics about players, games, and leagues. Examples include American football, basketball, hockey, soccer, and rugby.

The integration of a sensor system like ZXY and video, enables fine grained search like "Give me all sequences where TIL has three or more players insider the opposing team's 18-yard box". Using player positions, an automatic annotation could be developed which would further improve search capabilities. Events like corners, penalty kicks etc could be detected using the positioning data.

In [7], Johansen et. al. demonstrated how a system like DAVVI [30] could be used to build a content-based multimedia index using annotation derived from commentaries available on the Internet in a soccer scenario. One of their problems was that live commentaries are time coded relative to kick-off and that TV-productions often start several minutes before kick-off [7]. In ZXY's system, this is already recorded. Bagadus could benefit largely from such a video annotation system, and with integration with a system like DAVVI, Bagadus could search for video sequences of events like goals, corners, penalties and automatically create video sequences of those events.

One aspect that is important to keep in mind, is that some of the data ZXY acquire, like heart rate, are considered medical data which comes with restrictions and is not allowed to be release to the public.

### 5.2.2 Personalized streams

Several professional content providers deliver popular real-time soccer streaming services, like Schibsted's VG Live and TV2 Sumo in Norway. At best, these systems provide manually generated (static) video summaries of individual games.

In contrast to traditional broadcast scenarios, panoramic video enables the content consumer to manipulate the camera view direction and viewport size. In [31], Peter et. al. conducted an evaluation of distributing panoramic video sequences and visualizing panoramic video content on relatively low-end hardware, such as-top boxes and tablet devices. They presented a scalable solution for distributing panoramic video sequences to multiple viewers at high resolution and quality levels [31].

In our prototype, we have showed how we can use smart camera selection based on player tracking, to follow specific players and we have showed how we can create a panoramic view by stitching video together. This could be implemented into a streaming service, where users could change and manipulate the stream to their choosing.

### 5.2.3 Next generation sport streaming

In software architecture, *Publish–subscribe* is a messaging pattern where senders of messages, called publishers, do not program the messages to be sent directly to specific receivers, called subscribers. Instead, published messages are characterized into classes, without knowledge of what, if any, subscribers there may be.

In this section, we will shortly describe how a sensor system could be used to publish events to live stream viewers. In this scenario, we will imagine a streaming scenario, where you as a home user choose what to watch, depending on what is happening. In this example, we will use American football and "sunday night football" as a scenario.

"Sunday night football" is a term you would often hear when talking about American football. Most National Football league (NFL) matches are played on Sundays and often multiple matches are played at the same time. As a viewer, this forces you to choose which game you are gonna watch, and this could often be hard if you like multiple team and players.

A pub-sub systems could allow Bagadus to publish events as they are happening, where you as a viewer could subscribe to team X when they are attacking, or team Y when they are defending. Or if team X is 10 yards from the end zone and have a huge chance of scoring a touch down.

## 5.3 Other sports

In this thesis we have used soccer as our scenario, but we want to emphasize that our solution is applicable in any stadium sport. Some aspects, like following specific player, could be even be more relevant in other sports. American football for instance, is a sport where every player has a key role to play, in every play of the game. The TV broadcast mostly follows the ball and where it goes, however, if you are a young teenager aspiring to be Safety (a defensive position, often down the field), you might want to be watching your favorite player at that position.

## 5.4 Summary

In this chapter, we have described how Bagadus integrate many known components into a system which can be used in a sport analytic setting. We have also looked into some of the capabilities that local positioning sensors on sport athletes gives, when integrated with video and we have looked into the crystal ball, and suggested future applications such systems give.

# Chapter 6

# Conclusion

## 6.1 Summary

In this thesis, we have introduced a prototype demonstrating the capabilities of integrating a local positioning system, multiple synchronized cameras and annotation system in a soccer scenario.

Using many well known algorithms and techniques, we identified pitfalls and problem areas in terms of synchronizing multiple cameras, player tracking and video stitching. We have introduced a lightweight solution, in terms of processing, for tracking players in multiple cameras and also how player tracking can be used for smart camera selection in a camera array.

In contrast to the many soccer analysis tools of today, which relies on manual work and/or heavy processing, our approach is light-weight and almost fully automatic. We have showed that using annotated events by experts, we can create video sequences of those events. The video sequences could either be used by coaches as part of a soccer analysis tool, but we have also discussed how such a system could be used in a home entertainment setting. Finally, we have discussed and proposed applications that could benefit from our integration.

## 6.2 Main Contributions

On Alfheim stadium, we have installed cameras and developed video recording software that can record synchronized videos from multiple cameras. In addition, we have made a demo application that can play synchronized recorded videos, where the videos are rectified and de-

barreled in real-time – enabling us to track players in the videos. Player tracking is made possible, by using our ZXY database interface for delivering coordinates for players. We also introduced a video stitcher which is integrated into the application.

The novelty of this thesis lies in the integration of video, sensor network and annotation and the possibilities this integration gives. Through prototypes and demos, we have shown some of these possibilities and also discussed how this integration can be further utilized to create the next generation multimedia platform and soccer analysis tool.

## 6.3 Future work

In this thesis, we have showed how expert annotation can be used to create video sequences, however, much potential of lies in creating more searchable content and combining positioning data with annotated events. An example could be some kind scheme for automatically detecting interesting events like penalty kicks, throws etc. based on positioning of players.

One of our goals, was keeping all the components in our system handle four cameras in real time. Unfortunately, we did not find time increase the performance of our video stitcher the performance of it is far from real-time, unlike the rest of the components in our system. The stitcher algorithm, is essentially just moving and warping pixels from four images to one large images. Moving pixels is an embarrassingly parallel problem, making it well suited for off-loading to graphics processing unit.

For synchronized cameras, there are many aspects that we have not considered in this thesis. This includes for example areas such as photometric calibration of multiple cameras and exposure control in a camera array.

# Bibliography


[1] Interplay sports. http://www.interplay-sports.com/.

[2] Prozone. http://www.prozonesports.com/.

[3] Di Salvo Valter, Collins Adam, McNeill Barry, and Cardinale Marco. Validation of prozone: A new video-based performance analysis system. *International Journal of Performance Analysis in Sport (serial online)*, 6(1):108–119, June 2006.

[4] Stats Technology. http://www.sportvu.com/football.asp.

[5] ZXY Sport Tracking AS. Zxy sport tracking webpage @ONLINE, June 2012.

[6] Jürgen Assfalg, Marco Bertini, Carlo Colombo, Alberto Del Bimbo, and Walter Nunziati. Semantic annotation of soccer videos: automatic highlights identification. *Computer Vision and Image Understanding*, 92(2–3):285 – 305, 2003. <ce:title>Special Issue on Video Retrieval and Summarization</ce:title>.

[7] D. Johansen, H. Johansen, P. Halvorsen, B. Olstad, C. Gurrin, and C. Griwodz. Composing personalized video playouts using search. In *Multimedia and Expo (ICME), 2010 IEEE International Conference on*, pages 1534 –1539, july 2010.

[8] Alton L. Gilbert, Michael K. Giles, Gerald M. Flachs, Robert B. Rogers, and U Yee Hsun. A real-time video tracking system. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, PAMI-2(1):47 –56, jan. 1980.

[9] E. Trucco and K. Plakas. Video tracking: A concise survey. *Oceanic Engineering, IEEE Journal of*, 31(2):520 –529, april 2006.

[10] C. BenAbdelkader, R. Cutler, and L. Davis. Person identification using automatic height and stride estimation. In *Pattern Recognition, 2002. Proceedings. 16th International Conference on*, volume 4, pages 377 – 380 vol.4, 2002.

[11] Mark Everingham and Andrew Zisserman. Automated person identification in video. In Peter Enser, Yiannis Kompatsiaris, Noel O'Connor, Alan Smeaton, and Arnold Smeulders, editors, *Image and Video Retrieval*, volume 3115 of *Lecture Notes in Computer Science*, pages 1961–1961. Springer Berlin / Heidelberg, 2004.

[12] Aftenposten. Fotballfans blir stadig mer sofagriser @ONLINE, April 2012.

[13] D. E. Comer, David Gries, Michael C. Mulder, Allen Tucker, A. Joe Turner, and Paul R. Young. Computing as a discipline. *Commun. ACM*, 32(1):9–23, January 1989.

[14] Basler. http://www.baslerweb.com/products/ace.html?model=167.

[15] x264 Video encoder. http://www.videolan.org/developers/x264.html.

[16] FFmpeg. http://ffmpeg.org/.

[17] OpenJPEG. http://www.openjpeg.org/.

[18] Open Computer Vision. Open computer vision documentation @ONLINE, June 2012.

[19] Margaret Livingstone. *Vision and art: the biology of seeing*. Harry N. Abrams, New York, 2002.

[20] Charles Poynton. *Digital Video and HDTV Algorithms and Interfaces*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1 edition, 2003.

[21] H.264 : Advanced video coding for generic audiovisual services. http://www.itu.int/rec/T-REC-H.264.

[22] R. Gusella and S. Zatti. The accuracy of the clock synchronization achieved by tempo in berkeley unix 4.3bsd. *Software Engineering, IEEE Transactions on*, 15(7):847 –853, jul 1989.

[23] Gergely Vass and Tamás Perlaki. Applying and removing lens distortion in post production, 2003.

[24] Tom Watson, Eugene Johnson, Michael Gross, Bernard Frischer, Lisa Reilly, Michael Tuite, Brian Donovan Ken Stuart, Sarah Wells, and Barry Gross. Iath best practices guide to digital panoramic photography. http://www2.iath.virginia.edu/panorama/, july 2012.

[25] Parallax - wikipedia. http://en.wikipedia.org/wiki/Parallax, july 2012.

[26] Matthew Brown and David G. Lowe. Automatic panoramic image stitching using invariant features. 2007.

[27] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. "surf: Speeded up robust features. *Computer Vision and Image Understanding (CVIU)*, 110(3):346–359, 2008.

[28] Johan Gronvik. Precise video feedback through live annotation of football. Master's thesis, University of Tromsø, June 2012.

[29] Dag Johansen, Magnus Stenhaug, Roger B. A. Hansen, Agnar Christensen, and Per-Mathias Høgmo. Muithu: Smaller footprint, potentially larger imprint. july 2012.

[30] Dag Johansen, Håvard Johansen, Tjalve Aarflot, Joseph Hurley, Åge Kvalnes, Cathal Gurrin, Sorin Sav, Bjørn Olstad, Erik Aaberg, Tore Endestad, Haakon Riiser, Carsten Griwodz, and Pål Halvorsen. Davvi: A prototype for the next generation multimedia entertainment platform. pages 989 – 990, Beijing, China, october 2009. ACM International Multimedia Conference (ACM MM).

[31] Peter Quax, Panagiotis Issaris, Wouter Vanmontfort, and Wim Lamotte. Evaluation of distribution of panoramic video sequences in the explorative television project. In *NOSSDAV'12*, Toronto, Ontario, Canada, june 2012.