Autonomous football game tracking and recording via smart phone

Evgeny Fedko
Freie Universität Berlin
Bachelor Thesis

Supervision: Eugen Funk

Selbstständigkeitserklärung

Ich erkläre hiermit, das ich die vorliegende Arbeit selbstständig verfasst und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe. Weiterhin erkläre ich, eine Bachelorarbeit in diesem Studiengebiet erstmalig einzureichen.

Berlin, den 20.11.2014
Evgeny Fedko

Statement of authorship

I declare that I completed this thesis on my own and that information which has been directly or indirectly taken from other sources has been noted as such. Neither this nor a similar work has been presented to an examination committee.

Berlin, 20.11.2014
Evgeny Fedko

Abstract

There are thousands of football games played all over the world every day. Most of them are between amateur teams. Recording the games as videos for later analysis enables systematic performance improvements. But due to the lack of dedicated film makers and cameras the task is intractable for most of the teams and clubs. Thus, the idea of autonomous camera for football games emerged. The challenges are the small ball and the large distance of the camera from the field.

The presented approach relies on an algorithm implemented on a smart phone, which can automatically track and film the game. The system estimates the direction of the player motion on the pitch and rotates the phone via the attached motor accordingly. The assumption is that the direction where the most players move to, is the direction of the game progress.

The presented work is a framework incorporating multiple computer vision techniques such as corner detection, Optical Flow and robust filtering implemented on an Android smart phone. The technique estimates the overall player movement on the field and sends commands to the motor via a serial interface.

The player tracking algorithm was validated with test pictures and the system was tested on several football games of Berlin amateur teams. The results show, that the approach developed in this thesis can be used for autonomous football game tracking and filming using a standard smart phone.

Contents

Chapter 1

Introduction

1.1 Motivation

The game lasts 90 minutes, said the famous German coach Sepp Herberger [24]. But it is not the entire truth: the game itself is only a small part of football. To succeed, every team must train several times a week. This process involves not only physical training, but also the tactical studies. According to [2], the planning of training sessions undertaken by a coach plays a very important role when preparing a team to a game. There are lots of ways to make the life of a coach easier. One of them is the video-analysis system *Fubalytics*[1]. The system allows to analyze football games and to derive decisions for physical and tactical training sessions. One can upload a video of a game to the server and mark important fragments. The video snippets are augmented with meta information such as player references, the quality of the action, comments on the player performance etc. The results can be accessed, edited and watched at any time from every computer with internet access.

The key word here is *video*. It is neither easy nor cheap for some small clubs to have big professional cameras and committed people, who can film the games. However, using a smart phone as a recording device enables a huge amount of people to record games spontaneously and without to prepare any special equipment. Almost everyone has a phone or tablet with camera, so filming a game becomes much more convenient. But staying 90 minutes with a phone and following the game through its small screen - that is a little bit harder. One could assume, it could be easy just to rotate the camera, without looking at the screen. But it does not make the life easier, as it also requires constant attention. This motivated the research on an automatic

[1] `http://www.fubalytics.de/`

system, which can rotate the camera in the appropriate direction without human interaction.

An automated camera system is a great support for the whole football community enabling to film games and to use the results for training, analysis and social sharing. This also saves time and money, as they do not need to hire professionals and the club members can concentrate on something more exiting then filming a game. Moreover, for regular teams buying expensive equipment is not an option.

1.2 Research Question and Objectives

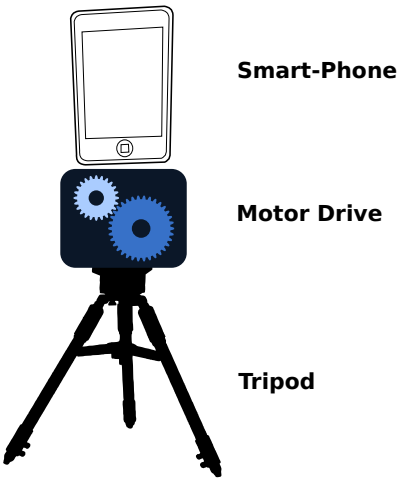


Figure 1.1: The concept illustrating the research objectives.

The main motivation of this research project is to enable automated recording of football games with a smart phone, since this usually includes a camera and in many cases a powerful processing unit. Figure 1.1 illustrates the concept of the system. Placing a smart phone on a tripod and connecting it to a motor enables recording and rotating of the camera at the same time without human intervention. This motivated the overarching
Research Question:

> *How can a smart phone be used to track and to record a football game automatically?*

In the first step, the system will need to estimate the movement direction of the players on the pitch using the image frames from the camera. In this step, we will use Android SDK to get the frames from the camera and a

C++ core for image analysis, which is the core point of our work. In order to get the direction of the overall player movement a robust algorithm is to be developed. The first research objective is particularly challenging, since the computation resources of the smart phone are very limited.
Research Objective 1:

> *Develop an efficient Computer Vision algorithm for tracking of the players in C++ and integrate it into an Android Java platform.*

If the movement is sufficiently strong, it will be necessary to rotate the camera. To do this, the system needs to send a signal to the Arduino motor control unit from the Android-App. The transferred information needs to contain the direction of the movement and the rotation duration. However, the communication between the app and the motor is challenging and needs to be specified and implemented in this work.
Research Objective 2:

> *Develop a communication and a motor control interface using Arduino and the Bluetooth protocol to rotate the camera while the game is being recorded.*

1.3 Thesis Structure

This thesis is structured in 5 Chapters. In the Chapter 2 related Computer Vision techniques are discussed with respect to the motion estimation task in the first research objective. The Chapter also examines the current challenges concerning object tracking from images.

Chapter 3 is devoted to the main contribution of this work which is the model of the player motion estimation and camera control: the main phases of data processing, related problems and implemented solutions are described. Each section of this Chapter describes an involved technical process: image extraction from camera, features and corner points detection, estimation of the motion direction with usage of statistical methods and estimation of hypothesis about the motion direction.

Chapter 4 provides a description of program components and its communication. Chapter 5 represents the evaluation result of motion direction estimation and answers the research questions posed in section 1.2. The final Chapter 6 summarizes the contributions and contains suggestions for future research directions.

Chapter 2

Related Work

2.1 General definitions

Computer Vision [18] is a field that includes methods for acquiring, processing, analyzing, and understanding images and, in general, high-dimensional data from the real world in order to produce numerical or symbolic information, e.g., in the forms of decisions. Computer Vision methods and its application will be the core of this work. The goal is to build the application by using existing Computer Vision approaches and by extending them in order to achieve the tracking of the overall player movement.

Another important part of our work is the Android operating system (OS) which is based on a Linux kernel. We have decided to use Android, because it is the most popular operating system for mobile devices in Germany: as for November 2013, 74.7% of all mobile devices in Germany have Android as an OS [19].

In the last decade, several approaches for automated object tracking in videos have been proposed. Orazio et al. [7] propose the algorithm for identifying the soccer ball using a modified version of Circle Hough Transform. They also deal with different lighting conditions, that could be useful in motion detection. But this approach is not fully applicable for this work because one cannot expect high resolution videos from a smart phone camera. Using low resolution videos the ball may be visible as a small area of a few pixels. Furthermore, circle detection via Hough Transform is very complex in terms of computation time. And as we have to deal with mobile devices which do not have much computation power, more lightweight techniques are required.

Budden et al. [5] propose a method, which is 300 times faster than the circular Hough transform approach. The idea is to determine the ball candidate in the context of a color-based vision system. This approach is also not applicable in this work: the distance between the ball and the camera must be less than 5 meters. In this case the algorithm determines the ball with less than 10% error. The higher the distance, the lower the quality of the detection. The use case of the system implies, that the distance between the ball and the camera can reach 100 meters or maybe even more.

It is necessary to find an approach, which does not deal with the ball itself, but with the overall player motion. Cheriyadat et al. [4] propose an object detection system that uses the locations of tracked low-level feature points as input, and produces a set of independent coherent motion regions as output. In order to achieve this, the Shi-Tomasi-Kanade detector is used to identify the low-level features. The low-level features are tracked over time using the Kanade-Lucas-Tomasi sparse optical flow algorithm. This is particularly relevant for this work because of the high computation efficiency. Furthermore, [4] deals with detecting of multiple regions of motion. So, we will need to extend the approach to detect only one relevant motion.

Thus, the analysis of the sparse moving features has been recognized as the most potential technique for the underlying problem.

2.2 Sparse Feature Tracking

The main idea is to analyze two consecutive frames of the video stream in order to determine the direction of the movement in the game. In order to achieve this, several related "low-level" steps need to be applied – foremost feature tracking and robust filtering for outlier removal.

The first step is to find good features to track. One approach to this problem was proposed by Shi and Tomasi in [17]. The key point here is to detect related *corners*. Corner detection is a technique, which allows to extract certain kinds of features and infer the contents of an image. The idea is to find such a point in the first frame so, that it would be easy to determine the same point in the next one. Choosing some white pixel on a white wall is an example of a bad corner. A good one would be to take some black pixels on the white wall. One of the first approaches was proposed by Harris and Stephens in [9]. This definition relies on the matrix of the second-order derivatives of the image intensities. We can think of the second-order derivatives of images, taken at all points in the image, as forming new "second-derivative images"

or, when combined together, a new Hessian image. This terminology comes from the Hessian matrix around a point. Corners, by Harris's definition, are places in the image where the autocorrelation matrix of the second derivatives has two large eigenvalues. Later Shi and Tomasi found out, that good corners resulted as long as the smaller of the two eigenvalues was greater than a minimum threshold. Shi and Tomasis method is not only sufficient but in many cases gives more satisfactory results than Harriss method, as reported by [3].

In our case we want to extract information from the sparse optical flow. Optical flow is the pattern of apparent motion of objects, surfaces, and edges in a visual scene caused by the relative motion between an observer (an eye or a camera) and the scene [22]. But the problem is that we do not have neither any prior knowledge about the content of these frames nor any predetermined objects to track. So, it is necessary to determine displacement, that represents the distance covered by the pixel. In order to do that, one can use the Lucas-Kanade method from [11]. This algorithm is very reliable and simple, because it relies only on some local information, derived from small area surrounding each point of interest. There are three main assumptions behind the Lucas-Kanade algorithm:

1. *Brightness constancy.* A pixel does not change its brightness when moving from frame to frame.

2. *Temporal persistence.* The image motion of a surface patch changes slowly in time.

3. *Spatial coherence.* Neighborhood points have similar motion.

Having found moving feature from two frames, the next step is to distinguish between two kinds of moving features:

- moving points caused by moving objects

- moving points due to the camera movement.

In order to determine the relevant moving points, one could use the Fundamental Matrix [10] – a relationship between any two images of the same scene that constrains where the projection of points from the scene can occur in both images. The fundamental matrix relates the points on the image plane of one camera in image coordinates to the points on the image plane of the other camera in image coordinates. Finding the features leading to a valid fundamental matrix, enables to exclude them as features from moving

objects. A method for detecting outliers using the Fundamental Matrix is proposed, e.g,in [20].

After the fundamental-matrix-filtering step one has only points that must be tracked. The last and the most demanding step is to determine the direction of the movement. The movement vectors have strong variations and a robust method is required to estimate a consistent movement direction. This is particularly challenging and the filtering problem is discussed for many decades in the statistics community. Some time ago, RANSAC [23] based approaches have been identified as very successful in removing outliers from imprecise data. Today, the approach is state of the art in Computer Vision problems such as estimation of a fundamental matrix [6] or robust surface extraction from scattered 3D points [15]. In the first step a model hypothesis is selected randomly from the given data set. After the model quality is estimated, the next hypothesis is extracted and also evaluated. After a finite number of trials, the model with the highest quality is selected and is considered to be optimal in the sense of the given quality metric. Given this idea, it is now required to develop a suitable quality metric to perform the task of overall player estimation from sparse optical flow features. As we do not have prior knowledge of the scale of inliers, we need to use some universal unbiased estimation method similar to approach described in [21], as we can not use standard statistical metrics.

2.3 Wireless Communication and Control

Blouetooth communication is being widely applied in many projects. The advantages are: the mobile device is not required to be physically connected with the communication counterpart, and the integrated hardware is widely available on all smart phone devices. The implementation of this communication is supported by the API of the Android OS – both for software and hardware part. Details about the implementation concepts using the Bluetooth protocol and an Arduino micro-computer for communication and control can be found in [14], where a small robotic car is controlled remotely.

2.4 Conclusion

In this Chapter we discussed the basic steps and concepts to provide the preliminary analysis of the video frames in order to extract the features to be tracked: optical flow, KLT feature tracking, fundamental matrix, RANSAC-

based filtering, and mean shift. Next Chapter discusses the integration and extension of the concepts into this work resulting in an efficient framework to be executed on smart phones.

Chapter 3

Player-Motion Estimation and Camera Control

3.1 Basic Concepts

The motion estimating process consists of several phases, which are illustrated in Figure 3.1. The explanation of the data transferred between modules of the system are listed in Table 3.1. The first step **p1** is to get the frames from the mobile device camera. Two frames (the current and the "previous" ones) are passed to the process **p2** for further analysis. The second step **p2** determines corner features to be tracked. Then, having this features, it is possible to estimate the motion direction in **p3**, using statistical methods. When the motion direction is estimated, it is time to perform the last step **p4**: rotate the camera in the corresponding direction. All these steps are described in more details in the following parts of this Chapter.

Data	**Data Explanation**
d1	Data-Pointer to of current frame n and address of frame n-3
d2	List of good features to track
d3	Direction and duration of camera motion

Table 3.1: Explanation of the data from Figure 3.1.

3.2 Phase 1: Image Acquisition

Each frame from the camera is acquired using the Android API. In this step there was a choice between a standard android API and the one for

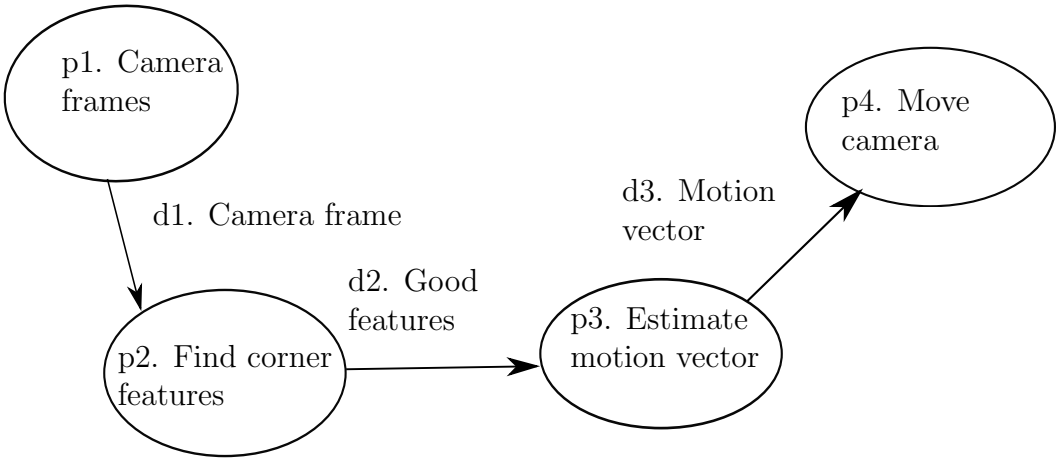


Figure 3.1: Workflow diagram of player-motion estimation and camera control.

OpenCV. On the one hand, the latter could fit better for the purposes of this thesis as it deals with the OpenCV library. But during the research and some preliminary experiments we have found out, that the OpenCV API for Android does not allow to save the captured video while the image analysis is running in a parallel thread. Thus, we focused on the standard Android API, which has a callback handler interface to access raw camera frames.

The next question was which frames need to be passed to the Computer Vision part of the application. Passing *every* frame could make the whole application very slow, as an Android device has not enough memory for complicated operations. So, in order to increase the efficiency, every third frame ($\triangle = 3$) is used for processing. The current frame i_t is stored in cache and is passed as previous $i_{i-\triangle}$ in the next iteration $\triangle$ frames later.

3.3 Phase 2: Finding Corner Features

The second process is the first core contribution of this work. It processes two consecutive camera frames and estimates the overall movement of the scene. The process returns a list of feature vectors which have been identified as not caused by the camera movement. Thus, only feature vectors representing the movement in the scene are extracted.

After extracting shift vectors, later referred to as *features*, between two images applying a sparse optical flow technique, the main task is to determine the corner ones, which represent moving objects, and to eliminate the "bad"

ones, which represent the camera rotation. This process is described in more detail in Figure 3.2. At every step we filter out irrelevant points so that at the end we get only the useful features that can be used for the estimation of the motion direction. Next, this processes is discussed in detail.

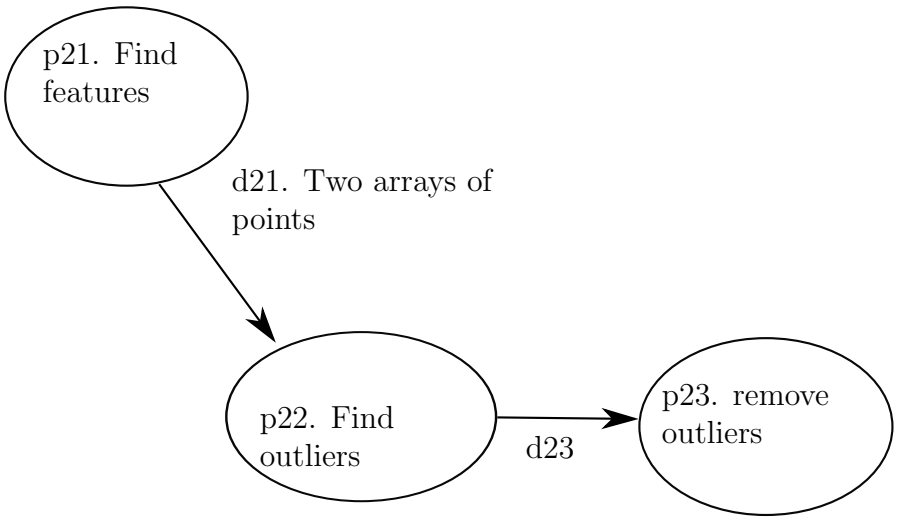


Figure 3.2: Workflow diagram of Computer Vision (**p2**) part from Figure 3.1

3.3.1 Phase 2.1: Finding Features

The Process of this stage is represented in Figure 3.2. The OpenCV part of the application gets two frames: the current one i_t and the "previous"[1] one $i_{t-\triangle}$. Next, good features to track are detected via Kanade-Lucas-Tomasi tracker technique from [12].

We evaluated, that the optimal number of features is 300: the hypothesis was, that all the necessary core features can be found in this case. This number (300) also combines two important aspects: on the one hand, it provides a sufficient level of quality, on the other hand – it is not too large, whereby it does not damage the real time processing.

First of all, we use the Shi and Tomasi method from [17], which computes the Hessian matrix and its eigenvalues at each image pixel. Pixels having strong variations in the eigenvalues are considered as corners and kept for tracking.

[1] As mentioned above, it is not a previous frame exactly, but the one spaced by three.

But this was not enough for the purposes of this work as we wanted to determine the features more precisely. To do this we used a Subpixel Corners Algorithm as described, for example, in [13]. It allows to get more resolution, than the simple pixel values supplied by Shi and Tomasi Algorithm. Loosely speaking, we need real-valued coordinates – not the integer ones, as we need more resolution for a better analysis. One might imagine needing to look for a sharp peak in image values. The peak's location will almost never be in the exact center of a camera pixel element. To do that a curve is fitted to the image values and then find where the peak occurred between the pixels. Subpixel corner locations are a common measurement used in camera calibration or when tracking to reconstruct the camera's path or the three-dimensional structure of a tracked object. See [3] for details.

We have now some points described as "good features to track". Now it is necessary to decide, how good these points are with respect to the application goal. The Lukas Kanade Algorithm takes two images and list of features to track. After all, the algorithm returns the points, that were tracked and the new locations of these points $\triangle$ frames later.

3.3.2 Phase 2.2: Finding Outliers

Another problem is to find out, which points belong to the motion itself and can be taken for the motion direction estimation, and which arise from the background changes. With other words, the players on the pitch move, and that is the motion of interest. But with the camera motion, some background points can be also taken for those moving. So, it is necessary to perform some action in order not to take such background-motion points in count. In order to perform the filtering, we use the fundamental matrix as a quality measure. This matrix contains the information about the translation and rotation, that relates two the cameras in physical space. It relates the points on the image plane of one camera in image coordinates (pixels) to the points on the image plane of the other camera frame in image coordinates, as shown in Figure 3.3.

This algorithm allows to find the outliers, which do not belong to the player movement itself.

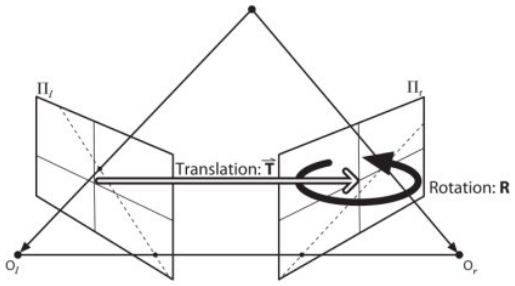


Figure 3.3: Fundamental matrix, picture from [3]

3.3.3 Phase 2.3: Removing Outliers

As the outliers are found in the Phase 2.2, it is possible to remove them from the further analysis. At the end of the Phase 2 it is expected to have only reliable movement vectors caused by the moving player and without the outliers.

3.4 Phase 3: Robust Movement Estimation

3.4.1 Statistical Filtering

In this step we calculate the hypothesis for the direction of movement. A set of reliable movement vectors is used as input and summarized to a single hypothesis vector. There is the following main hypothesis: the action happens in the same direction, so most of the motion vectors will point in the same direction. If we find the direction, with the most of the vectors point in, we will also find the direction of the motion and will be able to move the camera in this direction.

The first assumption is that the action always happens in the middle of an image. According to this assumption, it is necessary to delete those points, that lie close to the border of the image. So, all the tracked points, lying within of the 10-percent-zone of the border are not taken into account by further operations. As in Figure 3.4, the points from the black zone, lying within the 10-percent area of the border, are considered as unreliable for the motion estimation.

After the previous steps we have N movement vectors tracked by the sparse optical flow technique. First of all, it is necessary to determine the angles between every vector a_j and all the other $N-1$ vectors. To do that

Figure 3.4: Black zone points are not taken for the further analysis.

the angle between each movement feature vector is estimated via

$$\alpha_{ji} = \arccos\left(\frac{\bar{a}_j \cdot \bar{b}_i}{|\bar{a}_j| \cdot |\bar{b}_i|}\right). \tag{3.1}$$

At the end of this step, for every movement vector j there is a list of angles with all other remaining vectors. Now, it is necessary to determine the average angle for each a_j, using the arithmetic mean

$$\bar{\alpha}_j = \frac{1}{N}\sum_{i \neq j} \alpha_{ji} \tag{3.2}$$

Now we have an array of arithmetical means for every tracked point. The hypothesis is that the higher the value $\bar{\alpha}_j$, the more vectors point in a much different direction then a_j.

Let's assume, there are four vectors as shown in Figure 3.5. a_1 and a_2 have the similar direction and length, so they are supposed to represent the motion, whereas a_3 and a_4 differ from them in the aspect of direction and need to be filtered out.

To do that, one estimates the arithmetic mean for every angle with every other angle as described above with Equation 3.2. The result is four numbers, each of them representing the mean angle between the corresponding vector and all the other vectors. The next step is to calculate the mean value of all the mean values and the standard deviations s for each $\bar{\alpha}_j$. As we assume that the error in the movement vectors is normally distributed, we estimate the standard deviation over all $\bar{\alpha}_j$ and then filter out features with a too high $\bar{\alpha}_j$. In order to set the threshold for rejection, we compute the standard

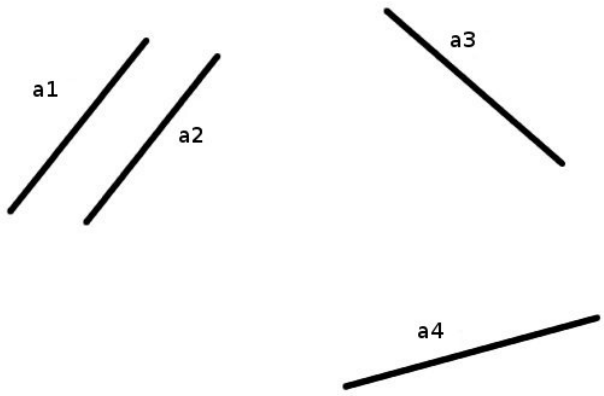


Figure 3.5: An example of the statistical filtering with four vectors. See text for details.

deviation [8]

$$SD(s) = s \cdot \frac{\Gamma\left(\frac{N-1}{2}\right)}{\Gamma(N/2)} \cdot \sqrt{\frac{N-1}{2} - \left(\frac{\Gamma(N/2)}{\Gamma\left(\frac{N-1}{2}\right)}\right)^2}, \tag{3.3}$$

of the sample standard deviation

$$s = \sqrt{\frac{1}{N-1} \sum_{i \neq j}^{N} (\bar{\alpha}_i - \mu_\alpha)^2} \tag{3.4}$$

with $\mu_\alpha = \frac{1}{N} \sum_j^N \bar{\alpha}_j$, N is the amount of samples and Γ is the gamma function [1] as an extension of the factorial function with its argument shifted down by 1. These calculations are based on the assumption, that $\bar{\alpha}_j$ follow the χ^2 distribution.

The behavior of (3.4) and (3.3) is illustrated in Figure 3.6 where the orange line represents s and the green one is unbiased estimator $SD(s)$. The y-axis represents the average standard deviation for every set of angles, and x-axis is just a number of every set.

Now, following the robust filtering approach, which is basically similar to the well known mean-shift approach, all elements a_j are deleted, which exhibit the standard deviation. As a result, there are only two vectors a_1

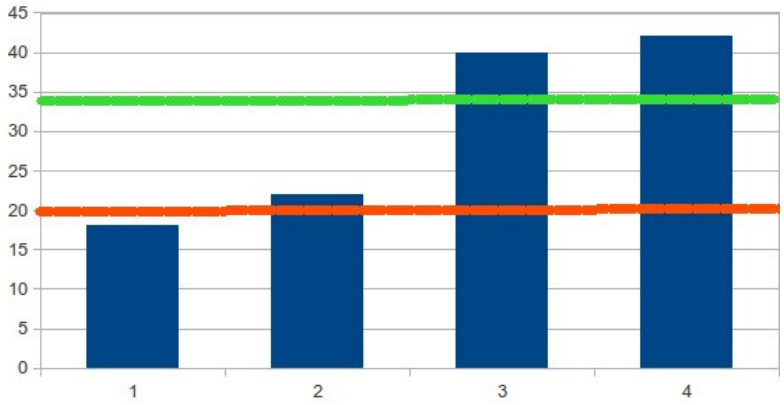


Figure 3.6: Sample standard deviation (3.4) (orange) and the overall standard deviation (3.3) (green).

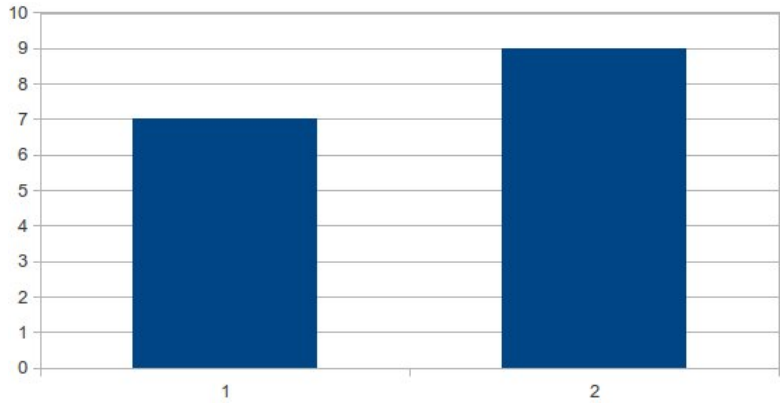


Figure 3.7: Only the first and the second vectors left after removing hypotheses with a too high standard deviation.

and a_2 as represented on Figure 3.7.

The next step is to find a vector, which has a minimal $\bar{\alpha}_j$. The hypothesis is that this point determines the vector, which correlates with the direction of the game movement. Finally, the motion vector of interest is computed as

$$\hat{a} = a_j | \alpha_j = \min_j \{\alpha_i\}, i \in \{0, \cdots N'\} \tag{3.5}$$

with N' as the number of vectors after the rejection described above.

3.4.2 Motor Control Interface

Next, it is necessary to decide, whether the camera should be rotated. If this is necessary, we then need to determine the direction and duration of this rotation. The most challenging here is to decide, how long the motor should be activated. This value must correlate with the distance between the image center and the estimated motion vector $\hat{a}$.

So, to rotate or not to rotate? First of all, we evaluate the length of the estimated motion vector $\hat{a}$, estimated in the previous step. If the length is more then $0.005 \times w$ with w as the width of the picture, then a signal in sent to the motor. From the previous step it is also possible to determine the direction of movement: Up, Down, Left and Right, which are the direction parameters of the motor. The simple protocol encodes the rotation direction and the duration in a single string, which is sent to the Arduino board via Bluetooth. To lower the camera view by activating the motor for 1 second, the generated string command is:

$$\{\text{D*1}\}$$

On the first place there can be one of four letters, corresponding to the direction of the movement: L(eft), R(ight), U(p), D(own). After the asterisk follows a number, which symbolizes the time to move the camera. It is also possible to use this protocol, if it is necessary to move in two directions. In this case the message looks as follows:

$$\{\text{LU*1\#2}\}$$

This could be translated as "Move left for 1 second and then Up for 2".

3.5 Phase 4: Moving the Camera

As last point the signal is sent to the Arduino control, which rotates the motor appropriately. The signal is sent via Bluetooth. A Bluetooth module is connected to Arduino board so that it can get data from the App. The Adruino board has a processor, which allows to program and then to perform some simple actions. In our case it has a program, that parses the command from the Phase 3 (sect. 3.4.2) and sends an appropriate signal to one of the four wires of the motor, which are connected to the board. Every wire corresponds to one of the four directions – left, right, up and down. The program sends an electrical signal to one of the wires for a certain period of time. During this period the motor turns in the corresponding direction. When the rotation is finished, the whole process starts again with the first Phase in Figure 3.1

3.6 Conclusion

In this Chapter the main principles of player-motion estimation and camera control have been discussed. These are acquiring images from the camera, good features detection, outliers detection, robust motion estimation and motor control. Next Chapter illustrates some implementation aspects. Chapter 5 presents the evaluation results of each of the discussed modules.

Chapter 4

Implementation

The core algorithm (phase 2-4) of the program is implemented in C++. The first phase and a small part of the fourth path are implemented in Java. In our application there are three big blocks: Android App, OpenCV part and Arduino. So, it was necessary to provide the communication between all of them. The whole interaction between components of the system is represented on Figure 4.1

4.1 Java Components

The first component is a simple standard Android API. But in order to process videos it was necessary to implement the picture rotation: when the device is rotated, the preview frame of the camera must turn smoothly and in the same direction.

Describing the structure of an Android App is not a goal of this thesis, so we will only say about the main functions used. All the communication between Java and C++ code takes place within the standard Android API function *onPreviewFrame*. It allows to perform own action on the camera frames. Here we take the current frame and the previous one, then submit them for the further analysis. The communication between Java and C++ is performed via JNI (Java Native Interface). It enables Java code to call and be called by native applications and libraries written in other languages – in our case in C++.

After all the C++ part returns the motion vector. The next task of the Java component is to pack it into the protocol (see Phase 4) and to send it

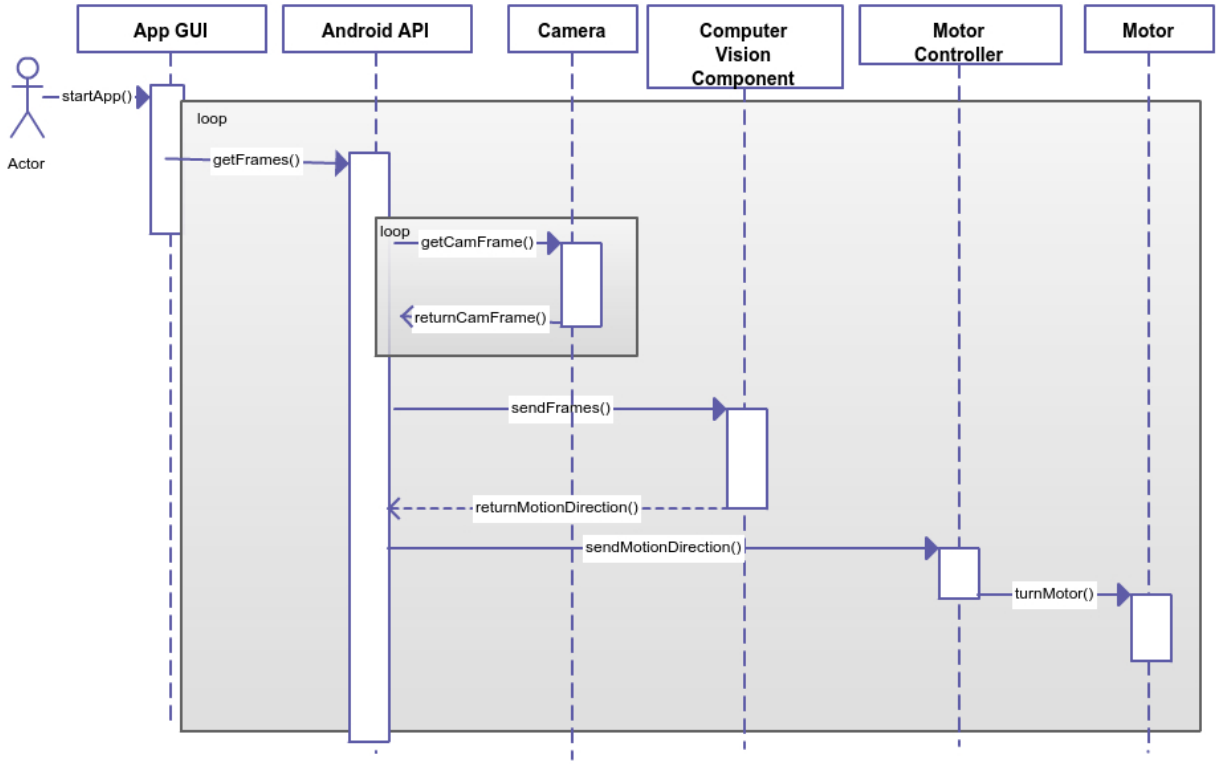


Figure 4.1: Components interaction scheme.

to the Arduino. The communication is performed via Bluetooth module.

4.2 C++ Core Algorithm

The main functionality of the application was implemented in C++ with the usage of OpenCV library. There were used the following core functions:

1. **cvGoodFeaturesToTrack**. This function implements the Shi and Tomasi algorithm. It takes two images to be analyzed. Another parameter is number of features to be tracked. In our case this value equals 100. The minimal acceptable lower eigenvalue for a point to be included as a corner is 0.01. And the minimal distance between two features is set to 5.0.

2. **cvFindCornerSubPix**. This function does subpixel corner finding. It takes the resulting points of the previous function as the initial guesses for the corner locations. The subpixel accuracy was set to three hundredths (0.03) of a pixel.

3. **cvCalcOpticalFlowPyrLK**. Computes Lucas-Kanade Optical Flow in a pyramid. The depth of the stack of images (image pyramid) is set to 5.

4. **findFundamentalMat**. This routine finds the fundamental matrix. As the method used in computing the fundamental matrix from the corresponding points, we used RANSAC-Algorithm (**FM_RANSAC** from OpenCV library).

We have also used some C++ data structures for images, such as IplImage and Mat. Furthermore there were implemented some other functions, necessary for the analysis:

1. **defineCosine**. Defines cosine between two given vectors.

2. **defineAverage**. Computes the average value for the angles between the vector given and all the other vectors.

3. **standrDeviation**. Computes standard deviation for an array of data.

... and some other small functions, that do not deserve to be mentioned in this text.

The Adruino program was also written in C++. It consists of a parser, that parses the incoming command, and of a component, that sends the corresponding signal to the motor.

4.3 All Used Frameworks and Components

All used tools and frameworks are listed below:

- **Android SDK** – API libraries and developer tools necessary to build, test, and debug apps for Android. We need to use it, because the main goal is to send signal from a mobile device to the motor, which will turn the device in the corresponding direction. Android App is the wrapper of the core motion estimation algorithm. It will get the camera frames and then also process them with C++ program running in background thread.

- **OpenCV for C++** – library for Computer Vision. Methods such as KLT Feature Tracking and fundamental matrix estimation will be implemented with the functions from this library. It also provides useful formats for working with images in different data types.

- **Java Native Interface (JNI)** – is a programming framework that enables Java code running in a Java Virtual Machine (JVM) to call and to be called by libraries written in other languages such as C, C++ and Assembler. As the main application is written in Java and the core functionality in C++, we need this wrapper to be able to call the C++-functions from within Java-code.

- **Arduino** – a single-board microcontroller (Figure 4.2). We send the direction of the estimated movement from the mobile device to Arduino. The board is connected to the motor, on which the smart phone is placed. Arduino receives the signal, interprets it and rotates the motor appropriately.

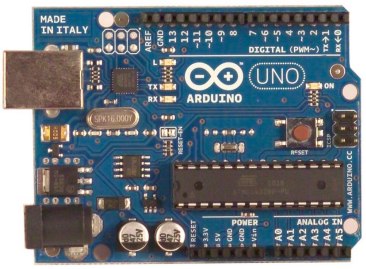


Figure 4.2: Arduino Uno board.

- **Bluetooth module** - RF Transceiver Module HC-06 RS232. It is not possible to send signal to the board through USB-wire, because a mobile device needs to be charged when filming for more then 90 minutes. To solve this Problem, we use a Bluetooth module, connected to the Arduino board.

- **Electronic Power Panner** - motorized camera host MP340 (Figure 4.3) is connected to the Arduino board. It can turn in four directions: left, right, up, down. The mobile device will be placed on this motor.

- All the diagrams for the Chapter Evaluation 5 are created with the means of the programming language **R** and with the plot-library for **Python**.

Figure 4.3: Electronic Power Panner.

4.4 Conclusion

In this Chapter the general program structure is described. It consists of two big parts determined with a programming language. Java component provides interface for retrieving the images. It also serves as communication interface between OpenCV component and the Arduino board. C++ components realize Computer Vision analysis of the motion. They include some core functions of the OpenCV library and also some other functions, that were implemented in order to perform the complete analysis. The program, running on Arduino board, is also written on C++.

Chapter 5

Evaluation

To make sure that the application really works we are going to use test pictures and paint all the tracked features on them. For example, we take two sequential frames of a video, Figure 5.1[1] [2].

(a) (b)

Figure 5.1: The first (a) and the second (b) images of the sequence

The core of the algorithm is statistical filtering as described in subsection 3.4.1. Now we are going to illustrate the work of this algorithm with real data – as example we take the pictures from Picture 5.1.

After filtering out the outliers with Fundamental matrix and also those lying close to the border, there is a set of vectors, which needs to be edited further. As described in subsection 3.4.1, we first calculate the average angle for every vector, then we calculate the mean value for all the average values and also standard deviations. As we have already mentioned in the previous sections, we are not sure, that the data are distributed normally. To check

[1]For the application we use all the pictures in grey scale. But here we use the colored ones just for clarity.

[2]All these pictures were processed not by the android-app, but by a C++ program applying the developed core algorithm in a test environment

this, one can use e.g. Kolmogorov-Smirnov test, as described in [16]. This test returns two parameters: D (absolute distance between samples) and p-value(probability of obtaining a test statistic result at least as extreme as the one that was actually observed, assuming that the null hypothesis is true). When testing the normality of the distribution, D must be smaller, then p-value. We performed this test, using the statistical programming language R. The test returned following values:

$$D = 0.9324, \text{ p-value} < 2.2e\text{-}16$$

That means, that the distribution is not normal. That is why we use approach of unbiased estimation for standard deviation, described in 3.4.1. We calculate the standard deviation for every angle, then the standard deviation of standard deviations and then the unbiased estimator to filter out next values. The result is to be seen on the Figure 5.2. The bars are the values of the standard deviations for every set of angles. As one can see, some of them lie over the green line, which represents the mean-shift: mean value plus unbiased estimator. Such bars and corresponding angles must be filtered out from the further observation.

The result of the outfiltering after the second iteration or the algorithm is represented on the Fugure 5.3.

As one can see, now we have much less bars, that are higher, then the green line. We need to filter them out and then proceed with the estimation of the motion direction as described in subsection 3.4.1.

For a more intuitive visualization, we are going to show the work of this part of the algorithm on an other concrete example. Lets assume, we have two frames of one video as presented on Figure 5.4.

The motion occurs from the right side to the left, and the most significant players here are the two in light blue form on left on top of the pictures. These players represent the motion of the game. Now we are going to illustrate the work of the algorithm on these two pictures. Figure 5.5 represents all the feature vectors before filtering out the outliers with the algorithm, discussed above in this Chapter. As we can see, we have a big variety of vectors, pointing in different directions.

Figure 5.6 represents the things after filtering out the outliers, which are shown as the blue lines. Now there are much less white lines, most of them are pointing in the same direction. The "average" motion (the red line) is

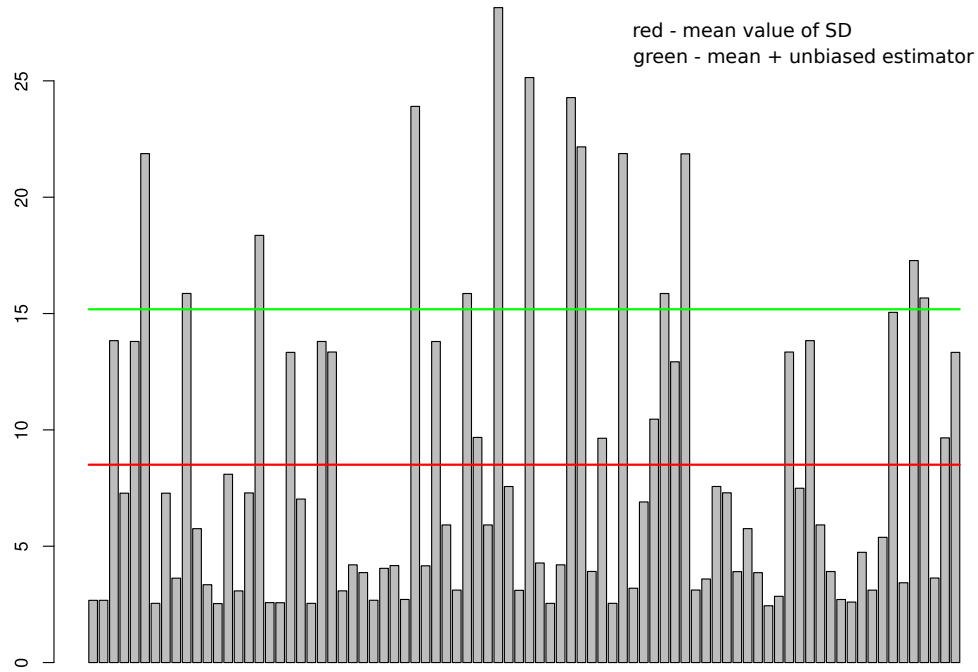


Figure 5.2: Statistical filtering with standard deviation of standard deviations and unbiased estimator.

also much more reliable, as on the Figure 5.5.

Lets go back to the example with the pictures, represented on the Figure 5.1. The result of all the algorithms used is to be seen on the Figure 5.7. This Figure is created as follows: the application gets pictures from Figure 5.1, performs all the necessary actions, and the result as well as the interim results get painted on the second picture. The **green** lines are the points, that belong to the fundamental matrix related movement. These are points, that appear because of the camera movement. The **orange** lines show the points, that lie close to the border and, according to the assumption, must not be considered as "good" features. The **blue** lines show the vectors, that are filtered out by the mean shift approach. The **white** ones show the good features to track and the **black** line is the mean position of the motion and its direction. The thick **red** line is the residue – the vector, that was chosen

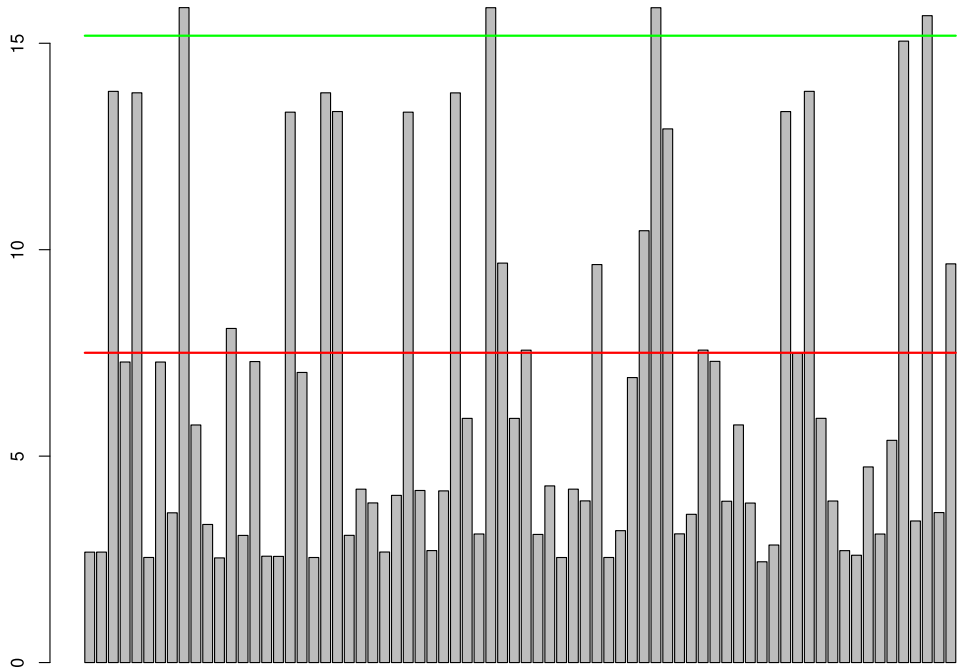


Figure 5.3: Statistical filtering with standard deviation of standard deviations and unbiased estimator. Second iteration.

as the best candidate, representing motion and its direction. The black one is the mean value of all vectors, that represent relevant features to track and have a small angle with the red one.

The work of the algorithm can also be illustrated with a sequence of frames from the real football video taken with the App as on Picture 5.8. This Picture represents the camera motion according to the events occurring on the pitch. As can be seen on two first pictures, the players are running from right to left. As it is seen on the next two pictures, the App reacts accordingly and turns itself to the left in order to be able to follow the game.

(a) (b)

Figure 5.4: The first (a) and the second (b) images of the sequence

Figure 5.5: Before filtering out outliers with the unbiased estimator. White lines – all tracked features

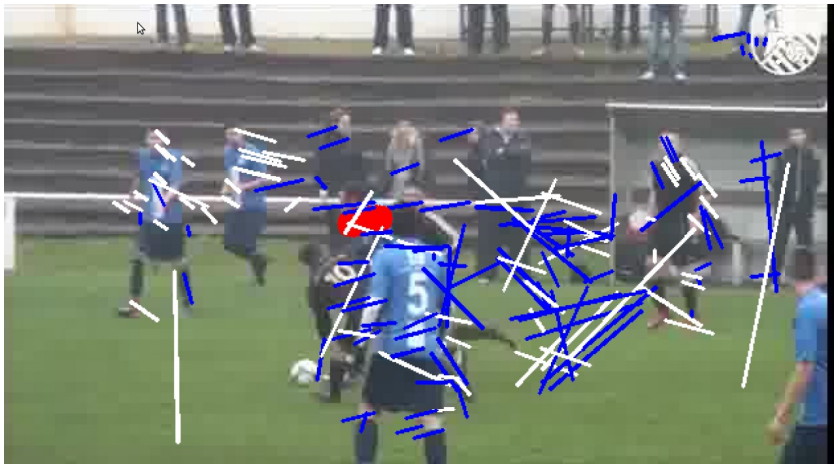


Figure 5.6: Before filtering out outliers with the unbiased estimator. White lines – all tracked features, the blue ones – outliers.

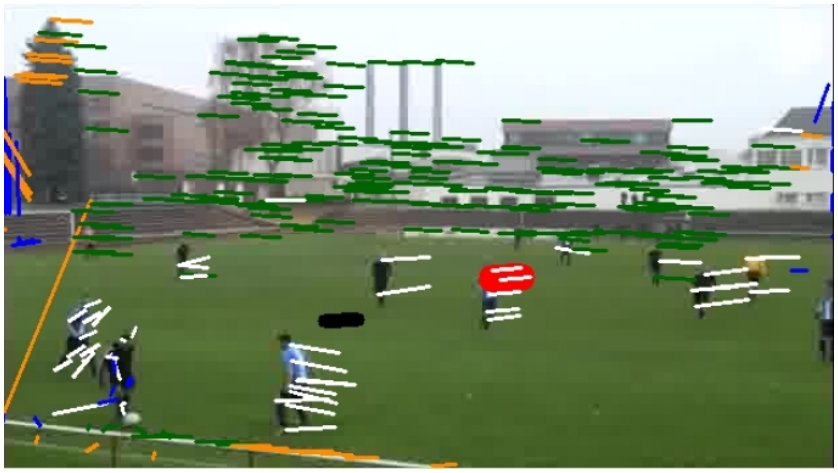
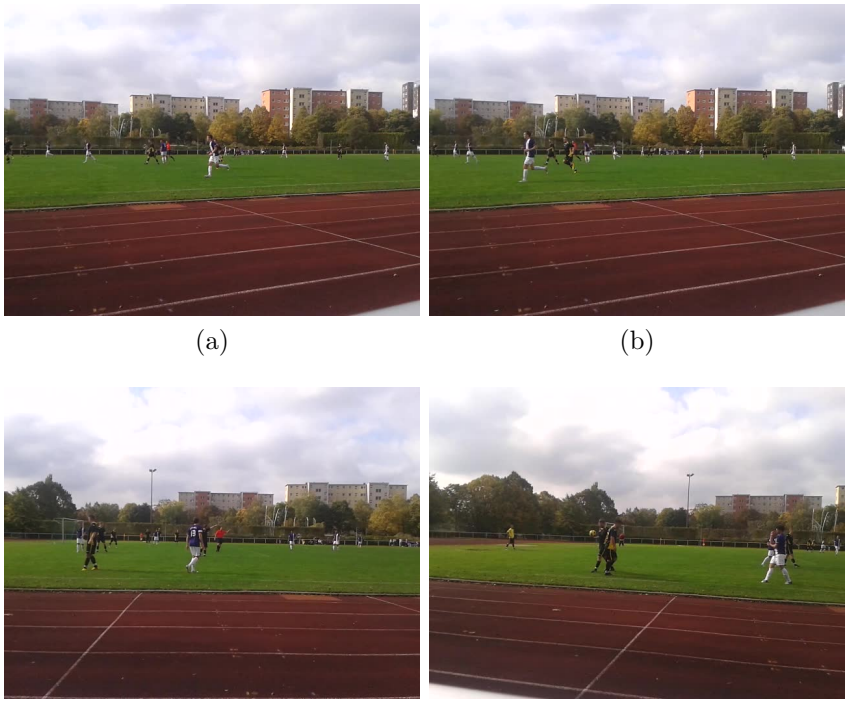


Figure 5.7: All the features and vectors, arising from the work of the algorithm. **Green** lines: fundamental matrix related movement. **Orange**: lying close to the border. **Blue**: filtered out by mean shift approach. **White**: good features to track. **Black**: mean position of motion direction. **Red**: the best candidate, representing the motion direction.



(a) (b)

(c) (d)

Figure 5.8: The four frames of a video sequence

Chapter 6

Conclusions and Future Direction

6.1 Summary

The study was set up to find a solution for an autonomous tracking of a football game and for implementing this solution as an Android application. The latter point is very important for small amateur football clubs, who can not afford to hire extra stuff for filming games, and also to buy some expensive cameras. The study sought to answer the following question:

- *How can a smart phone be used to track and to record a football game automatically?*

The main empirical findings are Chapter specific and were summarized within the respective empirical Chapters: Chapter 2 and 3. Based on this findings and also on related works, we have found an algorithm, which solves the problem. It was sequentially described in the previous Chapters.

As it is represented in Chapter 3, the images for the analysis are retrieved from a camera of a mobile device via Android App. The current image and the previous one are submitted for the further analysis. It is performed with Computer Vision techniques and functions, such as Shi and Tomasi method and KLT feature tracking in order to get the features to track. Then the outliers are detected – with means of fundamental matrix and some additional statistical metrics, such as standard deviation.

In this thesis we tested the application on the amateur football games played in Berlin. The main goal was to achieve autonomous camera motion,

so that it could track the game without human intervention. The examples of this process are represented in Chapter 5.

6.2 Lessons Learned

During this work we had to find solutions, to meet decisions and to solve some problems. All these skills and assumptions can be used in future studies. The most important ones are listed below.

- What can easily be done by a human, is not that easy to implement for a machine. There are lots of parameters on different levels influencing the correctness of the result.
- It is necessary to find a good way of ignoring the feature points of background: there are lots of them, they are not relevant, but at the same time they can strongly influence the final result.
- Techniques, good for a desktop computer or for a laptop, are not always appropriate for a mobile device – because of the lack of resources.
- There exists a possibility to combine code in different programming languages within one program. So, it is not necessary to "translate" it from language A to language B. In our case we used Java and C++ with help of JNI framework.
- Theoretical ideas need to be checked in practice. Only during this phase one could set all the necessary parameters and new functions, that would solve the problem more precisely.

6.3 Future Work

During the study we have achieved our goals, but we have also understood the directions, in which the next steps could be done in order to make the discoveries and tools work for some other areas of human live. These possible directions are briefly listed below:

- It could be useful to use the App for some other sport arts. It may seem, that the current App can already be used for these Purposes. But it is not so. Every sport has its special features – such as number of players, pitch size and the speed of game. All these parameters need to be taken in count when adapting the App for other sport arts.

- Another direction of development could be a better background processing. The App is made for amateur teams without lots of supporters around the pitch. But if the camera catches some fans, waving their arms, it could damage the result, as the motion vector could be determined wrong.

- It could be useful to track not the game, but some player during the whole game. It could allow clubs to analyze his or her action after the game without investing much money into operators and expensive cameras.

- Basically one could come up with many further use cases for such an App. The basic functionality of motion estimation can be with some small additions adapted to many other areas of human life and activities.

Bibliography

[1] St. Adams. An introduction to the gamma function. `http://warwickmaths.org/files/Gamma%20Function.pdf`, 2009. Accessed: 2014-10-30.

[2] Chris Anderson and David Sally. The Numbers Game. Why Everything You Know About Soccer is Wrong, 2013.

[3] Gary Bradski and Adrian Kaehler. Computer Vision with the OpenCV Library, 2008.

[4] Anil M. Cheriyadat, Budhendra L. Bhaduri, and Richard J. Radke. Detecting Multiple Moving Objects in Crowded Environments with Coherent Motion Regions. *Computer Vision and Pattern Recognition Workshops*, 2008.

[5] Josiah Walker David Budden, Shannon Fenn and Alexandre Mendes. A Novel Approach to Ball Detection for Humanoid Robot Soccer. *AI'12 Proceedings of the 25th Australasian joint conference on Advances in Artificial Intelligence*, 2012.

[6] R. Den Hollander and A. Hanjalic. A combined ransac-hough transform algorithm for fundamental matrix estimation. In *Proceedings of the British Machine Vision Conference*, pages 41.1–41.10. BMVA Press, 2007.

[7] T. DOrazio, N. Ancona, G. Cicirelli, and M. Nitti. A Ball Detection Algorithm for Real Soccer Image Sequences. *Pattern Recognition, 2002. Proceedings. 16th International Conference on (Volume:1)*, 2002.

[8] W.K. Härdle and L. Simar. *Applied Multivariate Statistical Analysis*. Springer, 2012.

[9] C. Harris and M. Stephens. A combined corner and edge detector. *Proceedings of the 4th Alvey Vision Conference (pp. 147151)*, 1998.

[10] R. Hartley and A. Zisserman. Multiple View Geometry in Computer Vision. *Cambridge University Press*, 2006.

[11] B. D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. *Proceedings of the 1981 DARPA Imaging Understanding Workshop (pp. 121130)*, 1981.

[12] Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. In Patrick J. Hayes, editor, *IJCAI*, pages 674–679. William Kaufmann, 1981.

[13] Luca Lucchese and Sanjit K. Mitra. Using saddle points for subpixel feature detection in camera calibration targets. In *Proceedings of the 2002 Asia Pacific Conference on Circuits and Systems*, pages 191–195, 2002.

[14] robotsrulz. BluetoothRC. `https://github.com/robotsrulz/BluetoothRC`, 2014. Accessed: 2014-09-12.

[15] Ruwen Schnabel. *Efficient Point-Cloud Processing with Primitive Shapes*. Dissertation, Universität Bonn, December 2010.

[16] J. David Sheshkin. *Computer Vision: Algorithms and Applications (Texts in Computer Science)*. Chapman and Hall/CRC, 2003.

[17] Jianbo Shi and Carlo Tomasi. Good Features to Track. *9th IEEE Conference on Computer Vision and Pattern Recognition*, 1994.

[18] Richard Szeliski. *Computer Vision: Algorithms and Applications (Texts in Computer Science)*. Springer, 2011 edition, October 2010.

[19] The Latino Post. `title=http://www.latinopost.com/articles/3045/20140111/ios-vs-android-market-share-2013-germany-google-os-conquer.htm`, 2013. Accessed: 2014-11-02.

[20] P. H. S. Torr and D. W. Murray. Outlier detection and motion segmentation. In *Proceedings of SPIE Conference on Sensor Fusion VI*, pages 432–443, 1993.

[21] Hanzi Wang and D. Suter. Robust adaptive-scale parametric model estimation for computer vision. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 26(11):1459–1474, Nov 2004.

[22] Wikipedia. Optical Flow — Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/wiki/Optical_flow, 2004. Accessed: 2014-10-30.

[23] Wikipedia. RANSAC — Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/wiki/RANSAC, 2004. Accessed: 2014-10-30.

[24] Wikipedia. Sepp Herberger — Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/wiki/Sepp_Herberger#Famous_quotes, 2004. Accessed: 2014-10-30.