Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Nizami Zamanov

Applying Computer Vision Methods on Mobile Devices for Ball Speed Measurements

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Faculty of Engineering and Computer Science
Department Information and Electrical Engineering

Nizami Zamanov

Applying Computer Vision Methods on Mobile Devices for Ball Speed Measurements

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Information Engineering
am Department Informations- und Elektrotechnik
der Fakultet Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prfer: Prof. Dr. -Ing. Marc Hensel
Zweitgutachter: Prof. Dr. rer. nat. Henning Dierks

Eingereicht am: 23. September 2019

Nizami Zamanov

Thema der Arbeit

Anwenden von Computer Vision-Methoden auf Mobilgeräten zur Messung der Ballgeschwindigkeit

Stichworte

OpenCV, Android, Objektverfolgung

Kurzzusammenfassung

Analyse und Implementierung von Computer Vision Methoden

Nizami Zamanov

Title of the paper

Applying Computer Vision Methods on Mobile Devices for Ball Speed Measurements

Keywords

Abstract

OpenCV, Android, Object Tracking Analysis and implementation of Computer Vision methods.

Contents

List of Tables

List of Figures

Listings

Acronyms

2D	Two dimensional. 5, 8
3D	Three dimensional. 4, 8, 15, 16
AI	Artificial Intelligence. 8, 9
CHT	Circular Hough Transform. 14, 36, 38
CV	Computer Vision. 1, 2, 5, 8, 9, 14, 16, 17, 21, 24, 26, 30, 32–34, 36, 48, 61
FA	The Football Association. 5, 27
fps	frames per second. 2, 7, 27, 29, 41, 42, 44, 48, 49, 61
HSV	hue, saturation and value. 6, 7, 9, 40, 56
HSVT	HSV threshold. 9, 45, 46, 48, 50, 56, 59
HT	Hough Transform. 36, 40
IDE	Integrated development environment. 33, 56
IFAB	The International Football Association Board. 4
IP	Image processing. 2, 8–10, 13, 26, 34, 36
km/h	kilometres per hour. 12, 17, 22, 24, 26, 29
LHT	Linear Hough Transform. 36–38, 45, 48, 59
m/s	metres per second. 12
ML	Machine Learning. 9, 32
OOI	Object of interest. 19, 36, 41, 57
RGB	red, green and blue. 6, 7, 56
SW	software. 1, 33
TD	Temporal difference. 48, 56, 57, 59

1 Introduction

Computer Vision (CV) is being used in a wide variety of real-world applications including medical imaging, autonomous driving, retail, manufacturing quality inspection, surveillance, object recognition, etc. Most of us came across it in everyday life, be it face detection system in cameras or image blending in applications like Instagram or Snapchat. With the advancements of digital cameras and popularity in the adoption of augmented reality experiences, the next decade is likely to see a considerable rise in the usage of CV techniques in software development.

Over recent years there has been an increasing demand for the use of analytics in sports [15]. It is of great interest from spectator, sportsman and coaching perspectives. One such application is a measurement of ball speed from a video recording. Such technology allows scoring in a game with actual physical achievement, as opposed to traditional video games. The core market for such applications, at least for now, are youth. But the potential market itself is huge, due to the fact that a considerable percentage of the world's population engages in some kind of sport and this number is growing together with raising awareness of people and trends towards a healthy lifestyle.

1.1 Motivation

Development in the mobile industry in the last decade has led to devices that are equally capable of personal computers a few years ago. They are getting equipped with better cameras and many sensors that are not usually found in their counterparts. On top of that, they offer mobility. All these factors make mobile devices very capable computers and feasible options for the development of computationally heavy software (SW) programs, such as real-time video analysis.

This project aims to develop software that measures the speed of the ball from a video recording. The benefits this could bring to the life of people training in football is great. There are already some devices commercially available for such tasks, but usually, they are not so affordable. Motivation to work on this project is listed, but not limited to the following:

- Mobility. Standard speed measuring devices do not fit into a pocket and people usually do not carry those devices with them.
- Affordability. Implementation of such an application on a mobile devices won't require additional resources, thus will cut the price practically to zero.
- Precision. As we are not targeting professional players, small errors in speed estimations should not be a big issue, especially considering the cost of purchase.
- Features. Professional speed devices are usually designed for a single task, measuring and displaying the speed. There is basically no infrastructure for diverse features. Mobile devices, on the other hand, offer internet connectivity, data storage and in general better environment for software development that might result in useful extra functionalities such as improvement tips, data storage, statistics, sharing, etc.

Last but not least, realizing the possibility that such an application could play some role in making smartphones usable for something else than information digestion is very compelling.

1.2 Objectives

The objective of this thesis project is to create a mobile application that makes it possible to measure the speed of a soccer ball from a video recording. The focus will be made mainly on the usage of such products during training sessions of kids under ten years old.

Furthermore, this thesis will research the capability of mobile devices in combination with CV methods for implementing such tasks. Some potential difficulties that need to be tackled in solving this problem are the fastball speeds and relatively low fps rate of regular smartphone cameras.

1.3 Structure

This paper is divided into nine chapters. Chapter 2 gives a brief overview of the terminology and concepts used throughout the project. Chapter 3 researches the related work in a subject topic and discovers existing solutions in the market. Next is Chapter 4, where requirements and stakeholders are defined, followed by Chapter 5, where solution concepts analyzed and selections on technology stack made. Chapter 6 examines different CV and Image processing (IP) algorithms for solving the problem. In Chapter 7 all previous findings put together into a more detailed solution and then implemented in Chapter 8. The results are evaluated and conclusions made in the last chapter.

2 Background

We start this chapter by defining the terminology used in this paper. We then provide all necessary background for soccer, digital cameras and mathematical equations used in this paper. The reader is also familiarized with the computer vision field and image processing algorithms.

2.1 Definitions

Product In the context of this bachelor thesis, the product refers to a system that is being developed throughout the paper.

Speed Gun Speed Gun (also Radar Gun or Speed Radar Gun) refers to a device used to measure the speed of moving objects. In this paper these phrases used for referring to classical devices exploiting the Doppler effect for measurements.

Soccer Since the word "football" refers to different sports in different countries, the word "soccer" will be used throughout this paper to avoid ambiguity.

Pitch Soccer field or any other surface where the soccer is played.

Goal Goal comprises two vertical goalposts on the sides that are linked with each other with a crossbar on the top and with a goal-line on the bottom.

Goalpost Vertical side poles of the goal. Usually made from steel or aluminum and colored to white.

Crossbar Horizontal top post of the goal that connects left and right goalposts.

Goal line Horizontal line connecting goalposts from the bottom. It is usually a white painted line on the ground.

Start position Initial still position of a ball.

Target Any point on the surface created by the goal-line, goalposts and crossbar.

Distance The distance from the ball to the goal.

Ideal trajectory The shortest distance from the ball to the goal, which is a straight line connecting the ball's initial position to the goal line's middle point.

Speed In this thesis, the speed is defined as the average speed which is the total distance traveled by ball divided by the elapsed time.

Scene Scene is the 3D world that is captured by camera recording.

Player The person who shoots the ball.

Operator The person who uses the product to measure the speed.

Detection The task of finding the position of a specific object in an image.

Tracking The task of finding or estimating the position of a specific object in a sequence of images.

2.2 Soccer

Ball According to The International Football Association Board (IFAB), soccer balls must have a spherical shape with a circumference between 70 cm and 68 cm [8, p. 43]. This leads to diameter values of 22.28 cm and 21.65 cm respectively, as per Equation 2.1, which describes the relation between circumference C and diameter D of a circle.

$$C = \pi \cdot D \tag{2.1}$$

Goal According to IFAB, the goal line must be 7.32 m long [8, p.35]. However, this is a dimension used in professional football. Different lengths are used for different age groups as illustrated in Figure 2.1. Since the focus of this thesis is to build a product for kids (Section 1.2), a *mini soccer* goal dimensions, 3.66 m long goal line and 1.83 m goal height, will be used where necessary.

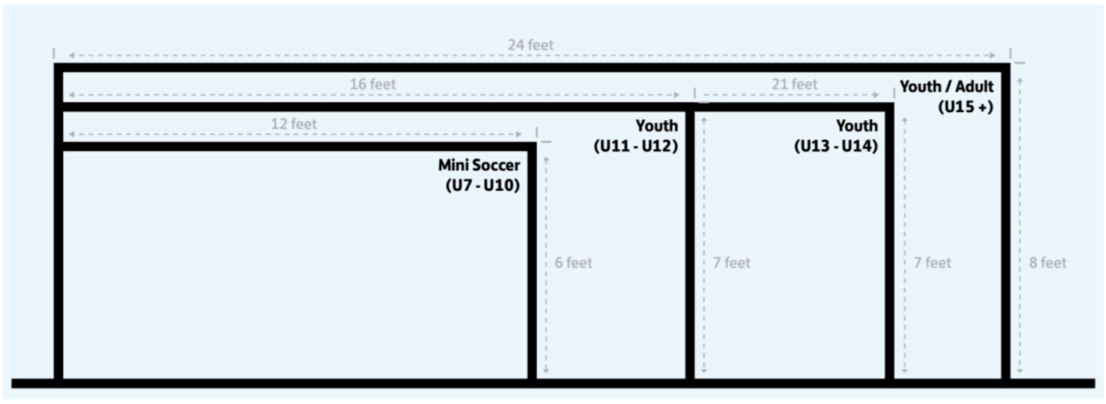


Figure 2.1: The goal dimensions for different age groups. Source: FA [7, p.6].

2.3 Digital Camera

2.3.1 Digital Image

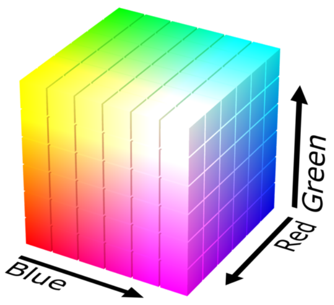
Digital image is a representation of 2D image in numerical form and it may be a raster or vector type. If the resolution of the image is not fixed it is of vector type. Otherwise, it is a raster image. Vector images can be scaled to any size without losing quality. Since most of the digital cameras make raster images, vector images are out of the scope of this paper.

Raster images made from finite set of numerical values, which are called *pixel elements* or shortly *pixels*. Pixel is the smallest individual element of an image. In essence, a raster image is a matrix of pixels with a certain number of rows and columns. The center of the pixel coordinate system is located on the top left corner. This corner is the first row and column with the index number 0. The image with 1080 x 720 resolution is said to have a width of 1080 pixels and a height of 720 pixels.

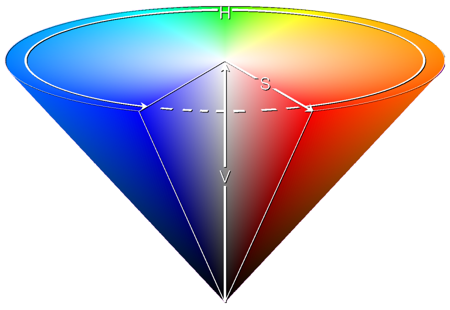
2.3.2 Color Spaces

Color space is a mathematical model to represent color information in different applications such as computer graphics, image processing, TV broadcasting, and CV [24]. Different color spaces are useful for different purposes. The next paragraphs introduce the ones that are relevant to the development of this work.

Binary Each pixel in binary images has only two values (colors); one and zero. They are also called *black and white* pictures.



(a) RGB



(b) HSV

Figure 2.2: RGB and HSV color spaces

Grayscale In grayscale images, each pixel is represented with a single value corresponding to the amount of intensity at that pixel. Hence, they are sometimes referred to as *intensity* images[29]. Grayscale images should not be confused with binary images. They have many shades of gray between black and white colors.

RGB An red, green and blue (RGB) image, also referred to as *truecolor* image, consists of three values per pixel, each representing red, green or blue intensity values (Figure 2.2a), or channels of an image. Red, green and blue are primary colors in RGB space. All other colors are derived from primary colors. The color of each pixel is determined by the combination of each three values at the pixel's location [29]. RGB color system is well suited for use in programming, as it is simple to manipulate and maps directly to the typical display hardware. However, it does not correspond to the human perception of color (e.g. doubling the value of blue color does not necessarily make the color twice as blue) [28, p.303]. Changing any color component modifies the color tone, saturation, and brightness at the same time. Therefore, color selection in RGB space is non-intuitive.

HSV Another color schema that describe the way colors combine to create the spectrum we see is hue, saturation and value (HSV) color space invented in 1970s [1]. The purpose of it is to make color selection more intuitive by aligning the digital representation of colors with the way human vision works.

In the HSV space colors are specified by the components *hue* (H), *saturation* (S) and *value* (V). It can be modeled as a cone as depicted in Figure 2.2b. Hue is the color portion of the

[1]Wikipedia, HSL and HSV, `https://en.wikipedia.org/wiki/HSL_and_HSV`, accessed: 07.08.2019

model, which can be defined as an angle in the range [0,2π] radians or [0,360] degrees [28, p.306-309]. It is a pure spectrum of colors. Saturation is the depth of color and measured as a distance from the central axis. In other words, it describes the amount of gray in each color. It varies between 0 and 100 percent. For a given hue value, if the saturation is changed from 0 to 100 percent, the perceived color changes from a shade of gray to the purest form of the color represented by its hue. The V component of HSV space describes the brightness of each color, also varying between 0 and 100 percent where 0 is black and 100 is the brightest version of a given color.

Contrary to extremely specific RGB values where it is hard to tell exactly how much of each primary color exists in a given pixel, HSV space provides better information about the color. In RGB space different shades of a color result in different values. In HSV space, however, the hue component stays similar through different shades of the same color. This aspect is helpful when searching for a specific color in an image.

2.3.3 Shutter Speed

Shutter speed is the length of time when the image sensor of the camera is exposed to light. It has a great impact on the appearance of images, especially when the scene involves an object in motion. Shutter speed controls the level of blur. High shutter speed can be used to freeze fast-moving objects, such as when recording a soccer shot. This permits to capture the moving object with less blur. Alternatively, low shutter speed makes the moving object, in our case the ball, appear blurred and leaving a long tail behind it.

2.3.4 Frame Rate and Video

Video is a sequence of images (or frames). The frame rate of the video is the number of images consecutively captured by camera per second. It is usually measured in frames per second (fps). Similarly, it can be expressed in hertz. One fps is equal to one hertz. The theoretical maximum fps is limited by the highest shutter speed of the camera.

This property of a camera is of great interest for the purposes of this thesis. Capturing a fast-moving object with low fps will create a video where an object appears to be jumping long distances between frames. Thus, very high fps values are beneficial for object tracking tasks. Although, high fps videos result in large video files, which consequently occupy a lot of storage and require lengthy processing.

Typical mobile device is expected to record a video at 25-30 fps. If we try to do real-time processing this value might shrink even more.

2.4 Computer Vision

CV tackles the problem of engineering artificial visual systems capable of somehow comprehending and interpreting our 3D world [28, p.3]. CV is an interdisciplinary scientific field, that tries to mimic the human visual system, so to say. It aims to create useful information from images or series of images. Different definitions have been given to CV; "the construction of explicit, meaningful descriptions of physical objects from images" [11], "computing properties of the 3D world from one or more digital images" [13].

Despite a lot of improvements in terms of new CV methods and powerful devices, there is still a long way to go for computers to interpret the images at the same level as humans. But why is it so difficult? In part because, as R.Szeliski describes in his book [27], CV is an inverse problem, where we are trying to rebuild the full 3D picture of the rich visual world from insufficient 2D information (image) that includes noise and other unwanted artifacts. On the other hand, it is difficult due to the desire to carry out a vast amount of CV computations in real-time.

The field is closely connected with other fields such as IP and Artificial Intelligence (AI). The next two sections explain the difference between them, followed by an introduction to some of the commonly used IP concepts.

2.4.1 Computer Vision vs Image Processing

The difference between the can be better understood by comparing their inputs and outputs. On both sides, we have some type of image file as an input. It is their output that makes them different.

In image processing, the ultimate goal is usually to enhance or otherwise alter the appearance of an image [28, p.2]. The output might be in different formats than the input, it might have resized, compressed, rotated or brightness, contrast, color space might have changed, edge detection filter might have applied. The key point is that in IP the main focus is not on extracting meaningful information about the content of an image. While CV on the other hand outputs some knowledge about the scene. Some of the popular topics in this field include scene understanding, object recognition, and tracking, and autonomous navigation [28, p.3]. In a way, IP acts like a tool for CV, helping to tackle its tasks. It is usually applied on the preliminary processing stage to prepare the input images for later use by CV algorithms.

2.4.2 Computer Vision vs Artificial Intelligence vs Machine Learning

AI can be understood as a term for any computer program that does something smarter than "basic" implementation of the lines of code. Some of the AI activities include speech recognition, face recognition, and decision making. In essence, AI tries to incorporate human intelligence into machines.

Machine Learning (ML) is a subset of AI and it is about incorporating computers with the ability to learn without being explicitly taught. Similar to IP and CV, ML can be thought of as a technique for realizing AI tasks. Usually, some data set is provided to ML algorithms to make machines learn themselves and later make accurate predictions based on those data sets.

To summarize, CV is a subset of AI which might use image processing or ML techniques to complete its tasks.

2.4.3 Thresholding

This technique is a simple and effective way of image segmentation. Thresholding can be applied to grayscale and color images. The output in both options is a binary image. Although there are different thresholding methods and styles [12, Chapter 4], for our purposes we are going to explore binary and HSV thresholding.

The idea behind binary thresholding is to check the intensity value of each pixel of a grayscale image and assign it to one value if it is greater than the threshold value (usually to one), otherwise to another value (usually to zero).

The idea behind HSV threshold (HSVT) is similar to binary thresholding. The difference is that now there are two, upper and lower thresholds, each a set of H, S and V values of HSV color space. Image pixels that fall in between the upper and lower bounds get the highest intensity value, while the others get the lowest.

2.4.4 Smoothing

In IP, smoothing (or blurring) is an attempt to capture important, major patterns in an image by reducing the details and influence of individual pixels. Thus, the output image is always a blurry. This method is also commonly used in the preliminary processing phase to simplify further image analysis.

Generally speaking, smoothing is equalizing the pixel values that are different than their neighboring pixels. It is usually done by the convolution of the original image with different filter masks. There are different types of blurring. The following are a selection of some of them that have been considered in the development of this project.

Normalized Box Filter This is the simplest blur filter. A convolution mask consists $N \times N$ unit matrix (N usually an odd number) divided by $\frac{1}{N^2}$. The result is the average off all the pixel values under the kernel area. The downside of this filter is that all neighboring pixels carry the same weight in the averaging. This might not be the best option in some cases, e.g. when preservation of edges is desired.

Gaussian Filter The Gaussian filter uses a Gaussian function (or Gaussian distribution in statistics) for filter coefficient weights. Instead of using the same values for all mask elements like in Normalized Box filter, the Gaussian filter puts a heavier weight on the center element and incrementally smaller wights on elements further from the center[12, p. 40-42]. As the name suggests, it is effective for removing Gaussian noise from the image. Although it performs better than the Normalized Box filter, edge preservation is still compromised.

Bilateral Filter Bilateral filter is one of the edge-preserving smoothing techniques. It smooths away noise while retaining sharp edges [2]. The bilateral filter is a function of space and intensity difference, while the Gaussian filter is only a function of space. This means that Bilateral filter, apart from the distance to nearby pixels, also takes into account the intensity difference between the pixel being processed and neighboring pixels. It ensures that only those pixels with intensities similar to a central pixels are included for the blurring. As explained in Section 2.4.5, neighboring pixels of an edge pixel, which do not represent an edge will exhibit large intensity variations to the central pixel being processed. The bilateral filter makes sure that those neighbors are not included for the computation. Hence edges retain their sharp intensity differences avoiding the averaging out by neighboring pixels' values [3].

2.4.5 Edge Detection

Edges play a dominant role in human vision system and detection of edges is one of the widely used operations in IP [28, p.121]. It is mostly used for image segmentation [4].

In digital images, edges are often signified by a sharp intensity changes along a particular orientation [28, p.122]. The simplest edge detection is based on first derivative and called *Gradient-Based* edge detection [28, Sec.6.2]. Taking the first derivative of an image with an

[2]OpenCV-Python Tutorials, Smoothing Images, `https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_filtering/py_filtering.html,accessed:31.08.2019`

[3]Wikipedia, Bilateral filter, `https://en.wikipedia.org/wiki/Bilateral_filter`, accessed: 31.08.2019

[4]MathWorks, Edge Detection, `https://www.mathworks.com/discovery/edge-detection.html`, accessed: 31.08.2019

edge in one direction results in a positive swing where the intensity rises and negative swing where it drops. A derivative of a multi-dimensional function taken along one of its axes is called partial derivative [28, Sec.6.2.1]. Taking the derivatives of an image in both horizontal (u) and vertical (v) directions gives us the gradient function. Based on the gradient function of an image the magnitude (strength) can be calculated at a given position (u, v). The local gradient of the image function is the basis of many edge detection operators. Prewitt and Sobel are two classic operators and practically, they only differ with the type of filter applied for estimating the gradient components and the way these components are combined [28, Sec.6.3]. Some edge detection operations are also interested in the edge direction angle and this information is also contained in the gradient function.

Whatever the operator is used, the result of previous edge detection operators are usually the values for edge strengths at each image position and possibly the orientation of the angle. The next step, in many situations, is picking each edge point and deciding whether a particular pixel is truly a part of an edge or not [28, Sec.6.4.3]. The simplest method is to apply a threshold.

Canny edge detector, published in 1986 by Australian computer scientist John Canny, is still state-of-the-art edge detection method [5]. Instead of applying a single threshold, the Canny operator applies two thresholds, a technique called "hysteresis thresholding". In its basic form, Canny operation uses the output of the Sobel operation. Following are the list of all steps in Canny operation[28, Sec.6.5]:

1. Smooth the image with filter of width σ.

2. Calculate the gradient vector in horizontal and vertical directions.

3. Determine the local gradient magnitude and orientation.

4. Preserve only those pixels that represent local maximum in the direction of gradient, that is perpendicular to the edge tangent. This is done with "non-maximum suppression" method. The result is edge points with 1 pixel width, which is more useful and desired information about edges.

5. Apply hysteresis thresholding. Two thresholds, t_h (high) and t_l (low) are defined. Each pixel of an image is scanned for an edge magnitude $E_m(u, v) \geq t_h$. Whenever such a pixel found, *edge trace* is started and all connected edge pixels (u', v') are added as long as $E_m \geq t_l$.

[5]Wikipedia, Edge detection, `https://en.wikipedia.org/wiki/Edge_detection#cite_note-11`, accessed: 31.08.2019

2.5 Mathematical Background

2.5.1 Speed

Italian physicist Galileo Galilei is usually credited with being the first to measure speed by considering the distance covered and the time it takes. The average speed v of an object is the distance d travelled divided by the time t taken [19]:

$$v = \frac{d}{t} \tag{2.2}$$

One of the common SI unit of speed is kilometres per hour (km/h). Many people perceive it better because in everyday usage they encounter it more than the other units. However, in this paper metres per second (m/s) is used for the computations since ball to goal distance is in meters and flight time of a ball typically happens in milliseconds or few seconds at most. Conversions between m/s and km/h can be carried out easily with the following equations:

$$km/h = 3.6 \cdot m/s \tag{2.3}$$

$$m/s = \frac{5}{18} \cdot km/h \tag{2.4}$$

2.5.2 Right Triangle

The right triangle is a triangle in which one of the angles is 90°. The sides a and b are adjacent to the right angle and are called legs (Figure 2.3). The side c opposite to the right angle is called hypotenuse. During conception phase in Section 5.2 knowledge on right triangles is required.

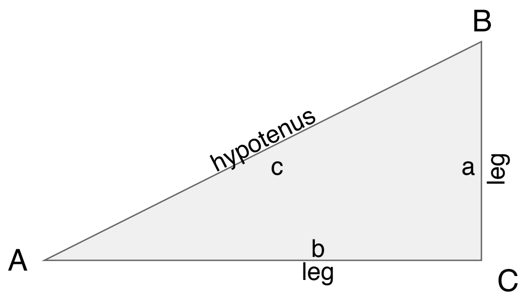


Figure 2.3: Right triangle

Pythagorean Theorem One of the first theorems of geometry that people learn is Pythagorean Theorem [26] which states that:

$$c^2 = a^2 + b^2 \tag{2.5}$$

Equation 2.5 will be used to find one of the missing sides when the other two sides are known.

Trigonometric functions Trigonometric functions relate an angle of a right-angled triangle to ratios of two side lengths. We are going to need the angle A, which can be found solving the equation $\sin A = \frac{a}{c}$ [3, p.6] to A:

$$A^\circ = \arcsin(\frac{a}{c}) \tag{2.6}$$

Equation 2.6 will be used to find the angle between hypotenuse and leg b.

2.5.3 Convolution

Convolution is a widely used technique in various areas of science [12, p.32-33]. In general, it combines two functions of the same dimensions to form a third function. It is extensively applied in IP during filter operations.

2.5.4 Kernel

Kernel (also mask, convolution matrix or filter) is a small matrix used in IP for blurring, sharpening, edge detection, and other tasks. It is essentially a fixed-size array of numerical coefficients along with an anchor point in that array, which is typically located at the center[6]. A kernel is convolved with the original image to achieve the results.

[6]OpenCV, Making your own linear filters, `https://docs.opencv.org/3.4/d4/dbd/tutorial_filter_2d.html`, accessed:31.08.2019

3 Related Work

There is several commercial and scientific work covering concepts on how a ball may be tracked. The review of the literature details these areas. Exploration of commercial products helps to identify existing CV techniques for soccer and other sports which would benefit from this research.

3.1 Literature Research

Several studies have been carried out for ball tracking using CV methods in soccer [4], basketball [9], volleyball [18], tennis [30, 32] and other ball sports. Some of them use high-quality cameras, some use multiple cameras, some use static cameras and others require the users to keep cameras as stable as possible. Based on my research, I have not come across a work that lets the user to shake the camera freely. Nevertheless, a review of the related literature should provide insights into the ball tracking and speed estimation tasks.

Pallavi et al. (2008) describe a method of detecting a football from broadcast soccer videos [4]. The key component in their video processing is a focus on ball trajectory. They employ Circular Hough Transform (CHT) to find ball candidates and then remove false positives using Optical Flow, motion and background subtraction methods. They successfully continue with ball trajectory estimation using dynamic programming.

Chakraborty and Meher (2013), in their paper "A real-time trajectory-based ball detection-and-tracking framework for basketball video", addressed the problem of ball detection-and-tracking in a real-time basketball video [9]. They discuss the challenges introduced by high speed and small size of the ball in relation to the video frame. They propose a solution based on the trajectory-based technique through the use of three frame differencing. Their system detects and tracks a ball using a two-fold method. First is feature-based and finds ball candidates based on shape and size. The second step verifies the detection by trajectory information by fitting ball candidates to a parabolic path, which is a typical ball path in a basketball game. This way they also fill in missing ball positions, due to occlusion and other reasons, using trajectory interpolation technique. Reportedly, their system performed well also in videos downloaded from the Internet.

Takahashi et al. (2016) developed a real-time volleyball tracking system using four cameras [18]. They draw attention to the difficulties in tracking fast-moving ball due to motion blur and occlusion by players. Therefore they developed a system that uses four HD cameras as sensors of ball position. The acquired data is processed in parallel, results integrated and 3D ball position is calculated. Thanks to the multi-camera system, if ball detection fails in the certain cameras, remaining cameras allow the system to continue normal work. This complementary tracking scheme constitutes the main characteristic of their system. The system's average error distance in 3D coordinates was 21.8 cm, which was about the diameter of a volleyball.

Many attempts have been made in tracking tennis balls, including Yu et al. (2004) [30] and Yan et al. (2005) [32]. They describe the problem of tracking the ball in a tennis video, which has many similarities with the problem of this work. With a lack of full control over the recording hardware and camera positioning, they are pointing out to difficulties including, abrupt ball trajectory direction change and ball blurring due to high speeds. The authors suggest avoiding object feature-based detection algorithms. Instead, they propose detecting all moving objects and evaluating their trajectories for candidate objects.

3.2 State Of The Art and Related Products

There are several systems already available in the market that resemble the one we are developing in this work. Some are being used for many years for traffic surveillance, others being developed lately to assist an objective decision making in sports and to produce analysis and statistics on match events. In the following sections, different commercially available speed measuring systems, including state-of-the-art and promising newcomers, are introduced. They include Doppler guns, multi-camera systems, and mobile applications.

3.2.1 Doppler Radar

Doppler radars or guns are used in law-enforcement and professional sports for fixing the speed of the vehicle, bowling speeds in cricket, tennis serves and etc.[1]. They exploit the Doppler effect to measure the speed of moving objects at a distance.

The Doppler effect is proposed by Austrian physicist Christian Doppler in 1842 [22]. This effect is based on the change in frequency of a wave in relation to an observer who is moving relative to the wave source. Most of us encountered this phenomenon in life when ambulance

[1]Wikipedia, Radar Gun, `https://en.wikipedia.org/wiki/Radar_gun`, accessed: 20.08.2019

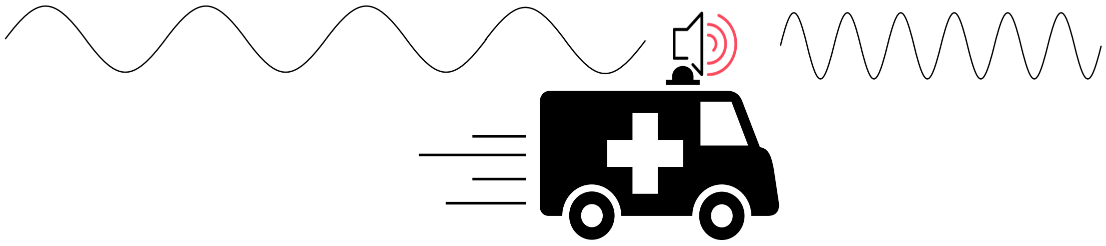


Figure 3.1: Demonstration of Doppler effect with high speed ambulance example.

sounding a sirens approaches us and then recedes. The received frequency during approach is higher than when it recedes (Figure 3.1).

Doppler radars consist of transmitter and receiver. The signal is sent out and then the same signal is received after it bounced off the target object. As mentioned above, due to the Doppler effect, there will be a frequency difference between transmitted and received signals as long as the target object is not stationary. Received signal will have a higher frequency than transmitted signal if the target object is approaching and vice versa. From that difference, Doppler gun calculates the speed of target object.

Doppler effect based speed measuring devices are so precise that they are also applied in measurements of the speed of planets, spacecrafts and even for estimating expansion speed of the universe[2].

3.2.2 Hawk-Eye

Hawk-Eye is a CV based system consisting of six or more high-performance cameras fixed around the scene that tries to create a 3D representation of the trajectory of the ball. Hawk-Eye is trusted as a fair second opinion by twenty different sports associations [2], that are using this system with individually calibrated and validated installations. An overview of the system is presented in Figure 3.2.

Hawk-Eye systems are based on the principles of triangulation, which refers to the process of determining a point in 3D space given its projections onto two or more images and timing data. It involves four major processing stages:

1. Detecting the center of a ball from each camera using CV.

2. Calculating 3D position of a ball using triangulation.

[2]NASA, S-4A-2 The Frequency Shift and the Expanding Universe, `https://www-spof.gsfc.nasa.gov/stargaze/Sun4Adop2.htm`, accessed 20.06.2019

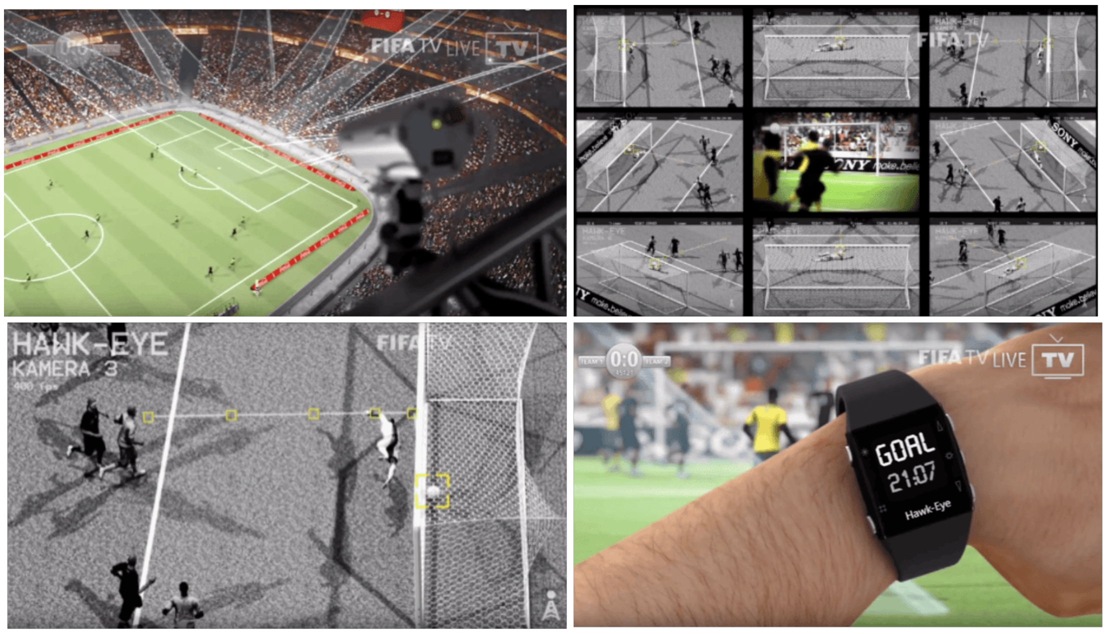


Figure 3.2: Hawk-Eye system overview

3. Repeating the second step for each frame to create a motion sequence.
4. Superimposing the trajectory over the court using virtual reality software to determine the position of the ball.

Hawk-Eye is becoming a standard in the growing number of sports and is a perfect example of what can be achieved by CV techniques. However, it is seemingly an expensive system requiring individual installation, that can usually be afforded by sport associations and established teams.

3.2.3 Athla Velocity

Velocity is an iOS-based speed camera application by Athla [3]. Athla claims that their application is the most accurate iOS speed camera. They prove their statement with a video demonstration on Youtube [4]. With this video they compare the Velocity to four different systems in the price range of 30-15,000 USD and interestingly, the application outperforms the most expensive system (PlaySigth) while remaining accurate within 0.5 km/h range compared to 1200 USD Doppler radar (Stalker), as shown in Figure 3.3.

[3]Athla, Main Page, `http://app.athla.com`, accessed:29.06.2019
[4]Youtube, Speed Radar "Battlefield!" PlaySight vs Stalker vs Bushnell vs SKLZ vs Athla, `https://www.youtube.com/watch?v=Khvj6tC1NPA`, accessed:29.06.2019

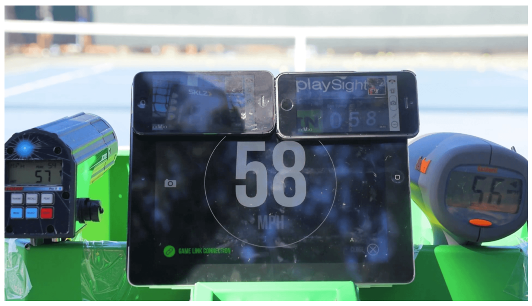
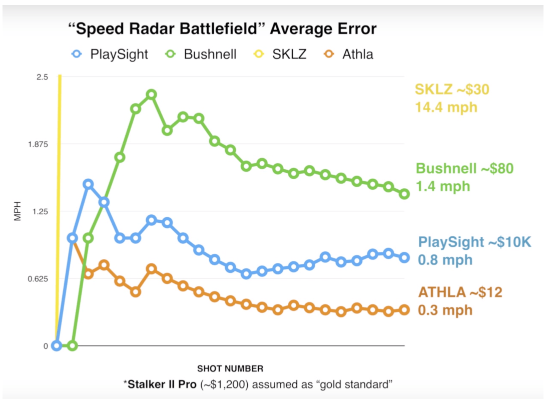



Figure 3.3: Comparison of Athla against other products. Setup (left) and measurement results (right).

In another video [5] the setup process of the application is explained. Since this application implements the speed measurement task similar to ours, it is of great interest and close investigation of this video might be helpful in getting an insight into the details of how the application with such a precision works. Setup process is as follows:

1. Choose the ball type (tennis, baseball, volleyball, soccer, cricket).
2. Set device 6 meters away and 6 meters to the side.
3. Place it on the ground
4. Place the device at 15-degree angle, front camera facing up.
5. Optionally, link to other devices (iWatch, iPad, iPhone) to see the results without having to check the screen of recording device meters away.
6. Measure.
7. If there is a problem, try to point it to an area of sky that is clear and is not covered by a lot of trees blowing in the wind.

Steps 4 and 7 suggest that their algorithms do not want to see anything else moving in the background apart from the ball. From Step 3 it becomes obvious that the algorithm requires static video shots which is probably one of the crucial points in this application's accurate estimations.

[5] Youtube, Athla Velocity iOS Speed Camera BASEBALL SETUP, `https://www.youtube.com/watch?v=tJ08IHetnbU`, accessed:29.06.2019

3.2.4 SpeedClock

SpeedClock is yet another mobile application that lets the user to measure the speed of cars, boats, animals, balls or practically any moving object whose dimensions are known [6].
The system is based on motion detection and object tracking. It has three different modes for measuring the speed[7]. One of the modes uses distance to target and angle information of the phone camera. In the next mode, the user enters the target object's size, such as the diameter of a ball, vehicle wheel or height of a person, and distance traveled by Object of interest (OOI) is derived based on this information. In the third mode, the user helps to detect fast-moving objects like a tennis balls by manually entering the target object's position once entering and leaving the scene.

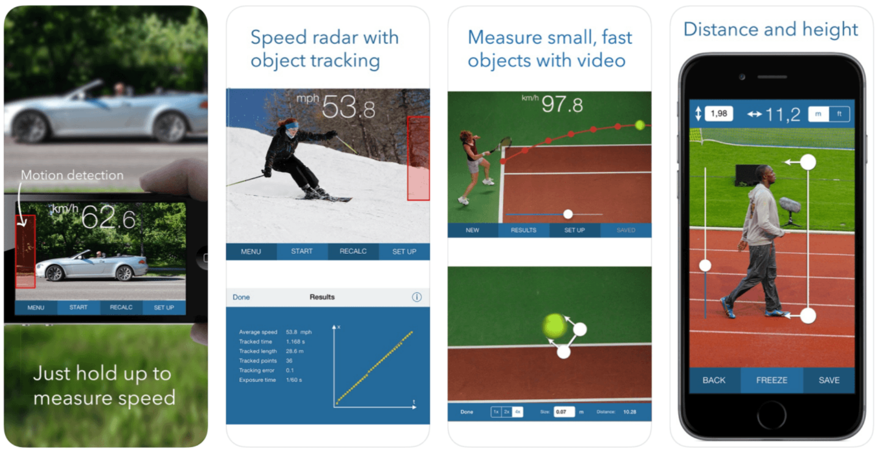

Motion detection
km/h 62.6
Just hold up to measure speed
Speed radar with object tracking
mph 53.8
Measure small, fast objects with video
km/h 97.8
Distance and height
1,98
11,2
BACK
FREEZE
SAVE

Figure 3.4: Screenshots of SpeedClock application

It performs good enough according to customer reviews on AppStore and Youtube, although no official or convincing test results were found. The downside of this application can be the fact that it requires a lot of pre-configuration before it can actually be used. Nevertheless, it is a promising tool when the user possesses the information about the surrounding object dimensions.

[6]App Store, SpeedClock Video Radar, `https://apps.apple.com/us/app/speedclock-video-radar/id400876654`, accessed:29.08.2019
[7]Appmaker, SpeedClock, `https://appmaker.se/home/speedclock/`, accessed:29.08.2019

4 Requirements Analysis

With this chapter, we start the development of the product by defining use-cases and identifying stakeholders. Afterward, we list the functional and non-functional requirements that define what a system is supposed to accomplish.

4.1 Stakeholders

Stakeholders of the product along with their priorities, rated from one to ten, are listed in Table 4.1. They can be categorized into internal and external groups.

Table 4.1: Stakeholders and their priority scores

Stakeholder	Priority
Supervisor	10
Secondary referee	9
Soccer coach	7
Soccer player	6
Individuals	5

Internal group represented by professors from the Hamburg University of Applied Sciences: primary supervisor for this thesis work, who is also the owner of this project's idea, and secondary supervisor who is also interested in the results of this paper acting as a referee. Intuitively, these stakeholders are getting first and second highest priority from my perspective, because it is these two stakeholders who will judge and define the success of this work. It is worth mentioning here that, this thesis work was conducted internally at the university.

The external group includes soccer coaches, trainees and any other perspective users that might use the product. Since this is not a commercial, client-driven product where customer and sale numbers come first, external stakeholders group is rated below the internal group. Individual users are getting the lowest score inside the group because one of the main focuses of this project is to build a product that will be used during soccer training by coaches and players. In a similar way, the soccer player gets lower priority than a coach since the ultimate end-user of the product, the operator, is a soccer coach.

4.2 Use Cases

In this section, the most important functionalities from the perspective of stakeholders, users and external systems are described.

The product being developed is essentially a radar system that estimates the speed of the ball from a video recording. Hence, first the foremost comes the functionality to measure the speed. The product is intended for the usage of soccer training, suggesting that there should be many trainees. It should be beneficial for the coach to create and save profiles of each trainee. Therefore, this turns into another functionality. The profiles can be stored in some sort of database and later be presented as helpful statistical data, making it yet another useful functionality.

When it comes to actors, we have the operator who is using the system. Analyzing the external systems, we should have a camera for recording, a database to store the data and a CV library that makes CV algorithms accessible [1].

All above information is summarized and depicted in Figure 4.1. The details of functionality "measure speed" is presented in Table 4.2.

4.3 Functional Requirements

Functional requirements derived from previous sections and received from stakeholders are listed in Table 4.3. They are enumerated with prefix "F" and will be referred throughout the text.

The first and foremost is the requirement F1, received from Supervisor. It states that the distance to be measured is 5-11 meters. Eleven meters match to the penalty distance in football [8, p.35]. Fortunately the distance has this limit, otherwise, the greater distance would require the operator to record a video further away from the ball trajectory, making a ball appear even smaller on the recording and thus making the ball detection hard.

The requirement F2 imposes extra task by demanding the readiness for shoots against a wall. As can be expected, a ball will rebound and change its initial direction after touching the wall. If not handled with caution, the ball tracking mechanism might be fooled, resulting in

[1]Depending on type of a device, camera and database might become an internal part of the system.

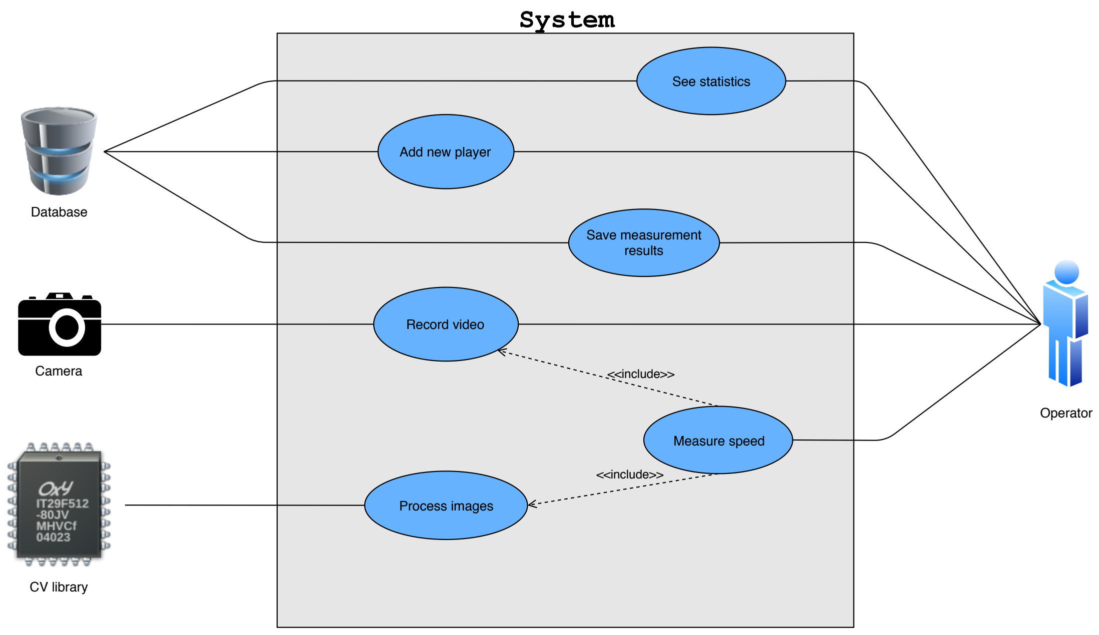


Figure 4.1: Use case diagram of the system

the ball-goal intersection moment to go unnoticed. Hence, this condition needs to be tackled accurately.

Requirements F3 through F7 declare functions regarding usability and extra features of the product. They impose extra data handling tasks. Among them, requirement F4 carries special weight because it will reduce the barrier to use the system.

4.4 Non-Functional Requirements

Non-functional requirements usually carries quality attributes of the system. They are listed in Table 4.4. Similar to functional requirements, they are enumerated with prefix "NF".

NF1 indicates ± 2.5 km/h error margin which is small enough to make the product usable. Bigger errors would make the product worthless because kids' shootings are not expected to cover a wide speed spectrum and greater errors won't give any sensible comparison of speed between hard and normal shot. This value is the same as for Supidio speed radar [21], the device which was used at the beginning of this work for supplying test data.

NF2 is a vital requirement about the performance due to its positive input to usability. In the modern world, users are used to instant responses from the software. If the user is asked to wait one minute before the result is available the product can be considered as a total failure.

Table 4.2: Use case table for "measure speed"

Use case	Measure speed
Description	Estimates the speed of ball from the video recording
Actors	Operator
Systems involved	CV library, Camera
Trigger	Operator presses the button
Preconditions	Ball and goalpost detected
Success end condition	Speed successfully calculated
Failure end condition	- ball has not reached the goal within specified time interval - calculation takes longer than specified time interval
Typical flow	1. Operator presses the button 2. Player shoots the ball 3. Ball reaches the goal 4. Speed is calculated based on available data

Table 4.3: Functional requirements

Id	Statement
F1	The system must measure the speed of ball when ball to goal distance is 5-11 meters.
F2	The system must support the measurement of shots against a goal and a wall.
F3	The user must be able to create and save new players to the system.
F4	The user must be able to measure the speed without choosing a player.
F5	The user must be able to save the measurement results to the player's history.
F6	The user must be able to sort statistics of all player by age, speed and date.
F7	The user must be able to filter statistics of all players by name, distance and team.

NF2 demands efficient algorithms and near real-time processing, yet giving enough room for post-processing.

As can be expected, not all shoots will follow the ideal ball trajectory, which is the shortest and predefined distance to the goal. After being shoot by the player, the actual ball trajectory can create horizontal and vertical (or both at the same time) angles with the ideal trajectory. Any other trajectory will have a greater distance than the ideal trajectory and thus needs to be considered. NF3 requires to increase the accuracy by eliminating the error caused by the vertical shooting angle. Luckily, there is no requirement for a horizontal angle which would put weight on performance, because it would probably require detection of changes in ball radius which in turn would require additional sets of CV algorithms and highly expensive computations.

NF4 and NF5 will also benefit the user experience. This requirement set puts a limit to the amount of external assistance, restricting the SpeedClock-like implementations (Page 19), wherein some modes user is asked to pause the video at different points and mark the position of the ball.

Requirements NF6 through NF8 dictates affordability concerns. Meaning, the solution must not require expensive, high-end devices. They will help in reaching out to the target user segment, taking into account prospective users who will most likely be teenagers and coaches, the segment with medium and low buying power. NF8 also increases the usability of the system, since carrying extra pieces of hardware is not something that the users love.

Table 4.4: Non-functional requirements

Id	Statement
NF1	The measurement error shall be within ± 2.5 km/h range.
NF2	The measurement result shall be available within 10 seconds after recording.
NF3	The influence of a vertical shooting angle should be taken into account.
NF4	User help for ball detection shall be limited to the beginning of a measurement.
NF5	User help for ball tracking and calculation shall not be required.
NF6	If solution is based on an Android device, version 6 and up shall be supported.
NF7	If solution is based on an iOS device, version 9 and up shall be supported.
NF8	If solution is based on a mobile device, no external hardware shall be required.

5 Conception

In this chapter, we conceptualize a solution that meets the requirements of the previous chapter. We make our assumptions and then continue developing theoretical, hardware and software solutions.

5.1 Assumptions

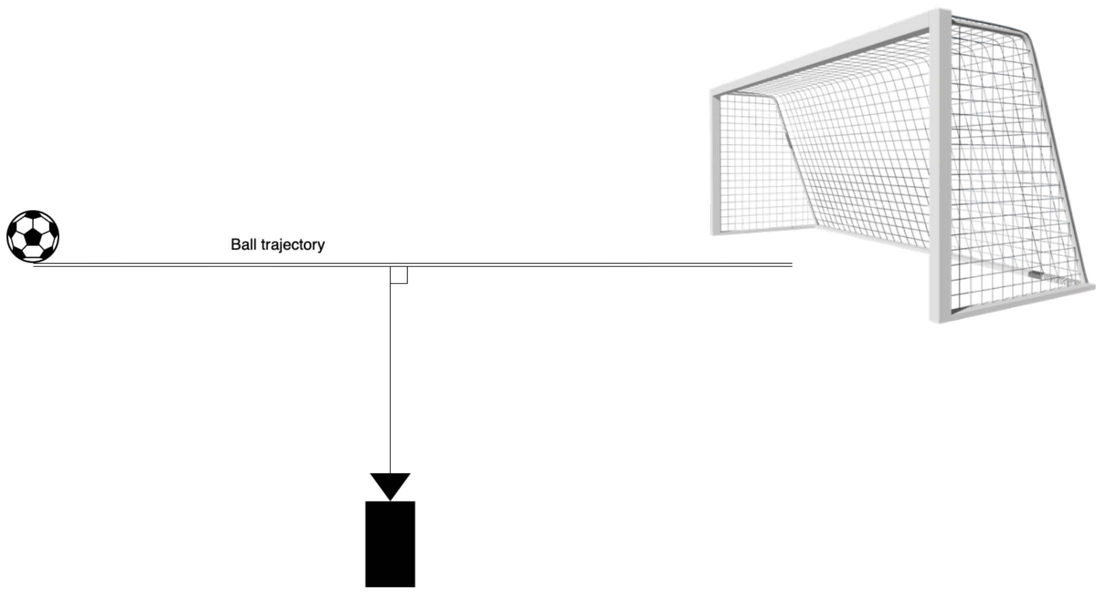


Figure 5.1: Camera lens direction with respect to ball trajectory

Direction Non-functional requirements list (Table 4.4) does not contain any requirement for considering horizontal angle. Thus, it is assumed that the camera will be positioned perpendicular to the ideal ball trajectory as shown in Figure 5.1.

Orientation The camera must be positioned in such a way that the x axis of the resulting frames from the video stays parallel to the ideal ball trajectory. This is necessary for identifying the ideal trajectory, which after this assumption, is any point between ball and goal that has the same height value as the ball's height value at its initial position. Ideal trajectory is needed for estimating vertical angle that appears between ideal and actual ball trajectories. Nevertheless, this assumption is only valid for the initial solution. In later versions of the product, it should be possible to derive the ideal trajectory by IP.

Stability The camera must be kept as stable as possible for accurate results. This assumption is also applicable to the initial solution. Future work could eliminate this assumption by applying image matching and registration techniques [28, p.565-587].

Lightening Videos can be recorded both indoors and outdoors as long as the ball's color is easily distinguishable from the background.

Background There should be as little motion as possible on the background of a ball trajectory.

5.2 Theoretical Solution and Failure Modes

Theoretically, finding the average speed is straightforward with the Equation 2.2 (Page 12). The values for distance d and time t needs to be known before. The distance can be supplied by a user or estimated using CV methods. The travel time of the ball from the start to the end can also be determined by capturing the system time at each frame. Next, the speed can be computed and later converted to km/h, due to better comprehension by humans. This approach can be implemented programmatically like in Listing 5.1.

```java
final long timeStart = System.currentTimeMillis();
// some code in between
final long timeEnd = System.currentTimeMillis();

final double travelSeconds = (timeEnd - timeStart) / (double) 1000;
final double speedInMps = distanceInMeter / travelSeconds;
final double speedInKph = speedInMps * 3.6;
```

Listing 5.1: Finding ball speed from time and distance

The solution described above is for best-case scenario when ball follows the ideal trajectory and the system detects the ball exactly at start and end [1] points. In reality, however, the ball

[1]the moment when 51% of the ball crosses the ball line

is expected to follow different trajectories and inherently limited fps rate of cameras might capture the ball a little after it crosses a goal line. Major failure modes have been found out, summarized in Table 5.1, then analyzed and tackled in the next paragraphs.

Table 5.1: Possible failure modes

Id	Description
FM1	Vertical angle
FM2	Horizontal angle
FM3	Vertical and Horizontal angle
FM4	Parabolic trajectory
FM5	Missed ball-goal intersection
FM6	Ball deflection
FM7	No ball-goal intersection

Vertical Angle (FM1) This is an angle created between the lines AB and AC (Figure 5.2). AB represents the ideal ball trajectory and its distance is known to vary between 5m and 11m, according to the requirement F1 (Page 23). BC represents the height of the goal which is also known to be 1.83m, according to FA specifications presented in Page 4. As can be seen from the figure, ABC creates a right triangle, allowing us to apply the Pythagorean Theorem (2.5). Solving the equation for c we are getting:

$$c = \sqrt{a^2 + b^2}$$

Inserting the values for a and b, and then applying Equation 2.6 from Page 13, results to lengths $AC_{5m} = 5.32m$, $AC_{11m} = 11.15m$, angles $CAB_{5m} = 20.1°, CAB_{11m} = 9.45°$ and ratios $\frac{AC_{5m}}{AB_{5m}} = 1.064$, $\frac{AC_{11m}}{AB_{11m}} = 1.014$, for AB values of 5 meters and 11 meters respectively.

Horizontal Angle (FM2) This is an angle created between the lines AB and AE. Although it is not part of the requirements to take this angle into account, it is worth exploring what kind of error this angle might introduce. Since BE is equal to BC and hence right triangle ABE has the same dimensions as the ABC, the values for lengths AE, angles BAE, ratios AE to AB will be identical to the ones found for vertical angle.

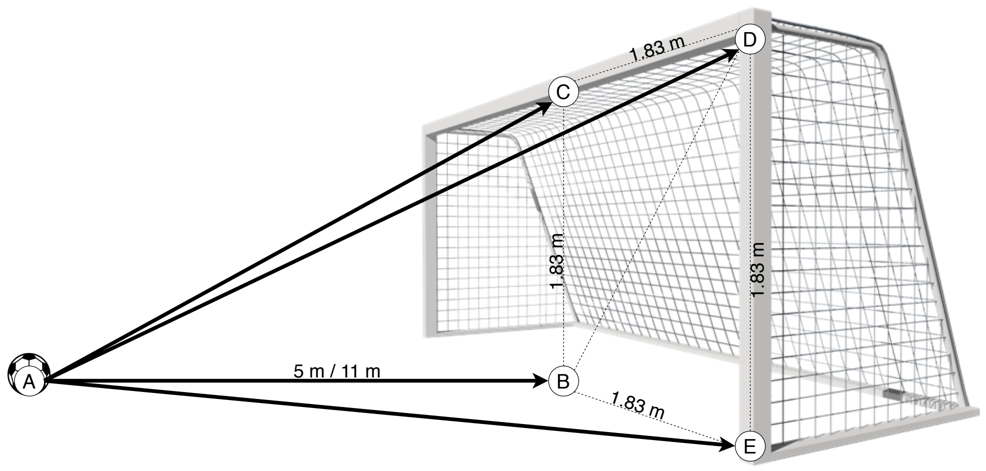


Figure 5.2: Ball and goal triangles

Vertical and Horizontal Angle (FM3) With the similar approach as for FM1 and FM2, following values calculated for triangle ABD: $BD = \sqrt{(BE)^2 + (ED)^2} \rightarrow BD = 2.59m$, lengths $AD_{5m} = 5.63m$, $AD_{11m} = 11.3$, angles $DAB_{5m} = 27.39°, DAB_{11m} = 13.25°$ and ratios $\frac{AD_{5m}}{AB_{5m}} = 1.126$, $\frac{AC_{11m}}{AB_{11m}} = 1.027$.

Parabolic Trajectory (FM4) Actual shoots in soccer tend to give the ball a parabolic trajectory in one or more directions. As a result, ball travels a longer distances than it would with a straight trajectory. Luckily, the trajectory of a ball in soccer does not usually create a sharp parabolic curve, which is typical to basketball and volleyball games. Thus, FM4 is not expected to be a source of big errors. Moreover, it is not part of the requirements.

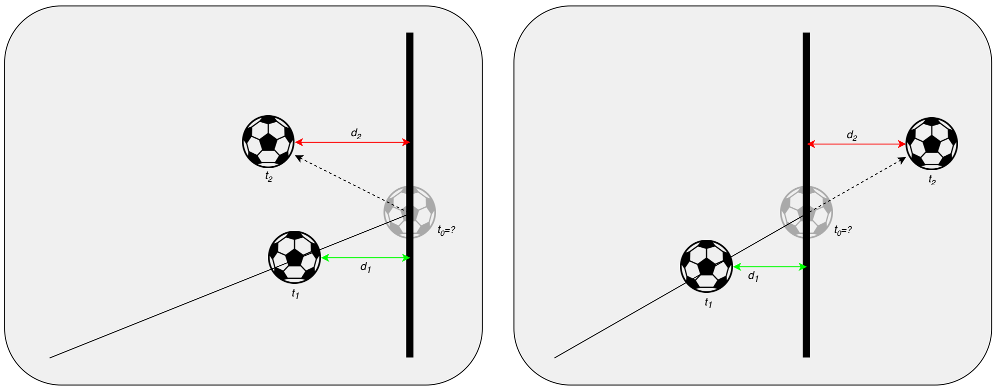


Figure 5.3: Failure modes FM6 (left) and FM5 (right)

Missed Intersection (FM5) FM5 stands for an event when a ball appears before the goalpost in one frame and further behind it on the next one. There is no frame in between showing the intersection moment of a ball with a goalpost. This caused by inherently slow camera fps rate. This mode will pose a risk for future algorithm that relies on intersection frame. As illustrated in Figure 5.3, time t_0 is unknown. It can be approximated as following, assuming stable ball speed and direction through all frames:

$$t_0 = t_1 + \frac{d_1}{d_1 + d_2} \cdot (t_2 - t_1) \tag{5.1}$$

Ball deflection (FM6) This model originates from the requirement F2 (Figure 4.3), which demands the measurement of shoots against a wall. Unsurprisingly, the ball should bounce back after hitting a wall. It can also happen after a ball hits the crossbar of a goal. It can be handled similar to FM5 and Equation 5.1, with some error caused by natural speed loss after ball deflection.

No Intersection (FM7) FM7 stands for an event when the ball gets shot but for some reason never appears to be reaching the goal. This might result in memory leaks in the system. To avoid that, the timeout should be set for the video recording phase. In general, the ball's flight time t_{flight} should not take more than two seconds, assuming fairly slow ball speed of 20 km/h and distance of 11 meters:

$$v = \frac{5}{18} \cdot 20km/h = 5.556m/s$$

$$t_{flight} = \frac{11m}{5.556m/s} = 1.98sec$$

Adding six more seconds as an extra margin for the player's preparation for shooting and user's delay in stopping the recording, a timeout can be safely set to eight seconds.

To obtain the speed errors caused by FM1 and FM2, test calculation was conducted for 5m and 11m distances, and for ball speeds of 60, 45, 30 km/h. As can be observed from Table 5.2, the error will be greater for shorter distances and higher ball speeds. The errors will be greater for FM3, since the angle created between ideal and actual ball trajectory is bigger. Nonetheless, as listed in the non-functional requirements table (Page 24), only the errors caused by the vertical angle (FM1) should be tackled.

Table 5.2: Errors caused by vertical or horizontal angles at different distances and speeds

Flight time (second)	0.3	0.4	0.6	0.66	0.88	1.32
Distance (meter)		5			11	
Speed (km/h)	60.000	45.000	30.000	60.000	45.000	30.000
Actual distance (meter)		5.32			11.15	
Actual speed (km/h)	63.840	47.880	31.920	60.818	45.614	30.409
Error (km/h)	3.840	2.880	1.920	0.818	0.614	0.409

5.3 Hardware

When it comes to hardware that is mobile and could potentially meet our requirements, two major options emerge: single board computers and mobile devices. This concern is addressed in the following sections.

5.3.1 Device Selection

A single board computer is a complete computer built on a single circuit board. Arduino Uno and RaspberryPi are examples of such devices.

Arduino Uno Arduino is a microcontroller that can't multitask activities and is best used for simple repetitive tasks. On the one hand, it is easy to setup and start prototyping. On the other hand, it is almost useless for expensive CV computations like in our case.

Raspberry Pi 3 Raspberry Pi is a single board, fully functional computer with its dedicated processor, memory and can run operating systems like Linux, Windows 10, Firefox OS and Android. It can run multiple programs at the same time and is faster than Arduino with 1.2 GHz CPU clock speed [23] compared to 16 MHz [6]. It can potentially use more than one video camera leading to robust object tracking and more accurate speed measurements. On the downside, users will have to buy an extra devices to use our product. Moreover, it will require more time to setup the development environment: operating system, external cameras, drivers, display and, etc.

Smartphones Android and iOS phones, two common mobile platforms, are very capable nowadays. In general, smartphones offer a great package: built-in cameras, multiple sensors, powerful processors, a great development environment with good documentation and high availability among people. All of these factors make smartphones a great fit for the purposes of this project, considering that the alternatives, namely RaspberryPi and Arduino, have many shortcomings.

To conclude, smartphones are just better candidates in terms of user-friendliness, affordability, computing power and ease of development. Most importantly, this option won't require the users to buy an extra dedicated device which would significantly reduce the chance of product ending up on the user hands.

5.3.2 Platform Selection

It is no secret that there are two major players in the phone market and mobile developers shall make a choice to go one way or the other. Fortunately, in recent years more and more tech companies are trying to solve this issue by offering programming languages and frameworks that eliminate this problem. Solutions are dozens of hybrid and native app frameworks[2] that allow to create both Android and iOS native codes from the single code base. Most promising hybrid frameworks from my perspective are ReactNative by Facebook and Flutter by Google. The advantage of such development would be writing one code and running them natively on both platforms. However, a hybrid approach might not go well with graphics-oriented, performance-centric apps [3]. It might still worth testing, but due to the time constraints it is not feasible. Therefore, the decision has to be made between Android and iOS.

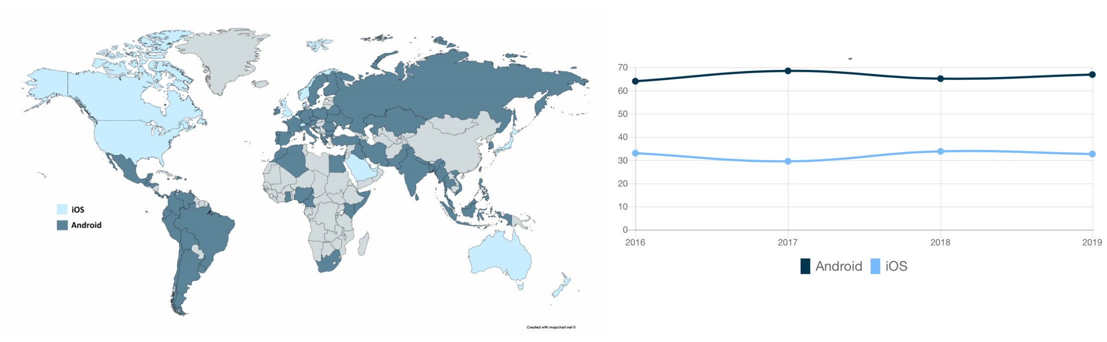


Figure 5.4: IOS and Android market share in the World (left) and Germany (right)

[2]Hackernoon, Top 10 Best Mobile App Development Frameworks in 2019–20, `https://hackernoon.com/top-10-best-mobile-app-development-frameworks-in-2019-612b95cf930f`, accessed:29.08.2019

[3]Medium, Choose the best — Native App vs Hybrid App, https://codeburst.io/native-app-or-hybrid-app-ca08e460df9, accessed:10.07.2019

Instead of doing an exhaustive comparison of every single aspect of these platforms (interested reader is welcomed to do so [4]), the focus is made on their market share. As illustrated in Figure 5.4, the coverage depends heavily on the country. People in countries like the USA, Canada, New Zealand, Australia prefer iOS devices while almost the rest of the world prefers Android devices. Developing a product primarily for German users, let's take a closer look at statistics in Germany. Figure 5.4 clearly indicates that about 70% of the German population prefers Android devices. This is almost two times more than its counterpart. Therefore, the product will be based on a mobile device that runs on the Android operating system.

5.4 Software

In this section we will try to find suitable CV library for further development. Apart from that, selections on other aspects of the development will also be defined. and programmatic approach for the development.

5.4.1 Computer Vision Library Selection

CV tools have evolved over the last years in such a rate that apart from software libraries, computer vision is now also being offered as a service by major cloud service providers such as Google, Microsoft, and Amazon. In this part, the most popular alternatives are evaluated.

TensorFlow TensorFlow by Google is an end-to-end open source ML platform [5] based on deep learning neural networks [6] used across range of tasks: classification, prediction, image recognition, motion detection, etc. [7]. One of the main drawbacks of TensorFlow in that it is extremely resource hungry. Moreover, to use it one needs to have experience with ML and Deep Learning. Therefore this option was not feasible for me.

OpenCV OpenCV is an open-source CV and ML software library employed by many well-established companies like Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda, Toyota. It is originally written in C++. It has Python, Java, Matlab interfaces and supports Windows, Linux, Mac OS, and Android operating systems. The library has more than 2500 optimized

[4] Diffen, Android vs. iOS, https://www.diffen.com/difference/Android_vs_iOS, accessed: 10.07.2019
[5]Tensorflow, About, https://www.tensorflow.org, accessed:31.07.2019
[6]Wikipedia, TensorFlow, https://en.wikipedia.org/wiki/TensorFlow, accessed:31.07.2019
[7]Exastax, Top five use cases of TensorFlow, https://www.exastax.com/deep-learning/top-five-use-cases-of-tensorflow/, accessed:31.07.2019

algorithms[8]. These set of classic and state-of-the-art algorithms can be used for a full range of CV tasks including but limited to face recognition, object identification, camera movement tracking, moving object tracking and 3D model extracting. All these factors make OpenCV a promising candidate and a very compelling library to employ.

BoofCV BoofCV is relatively young CV library. While OpenCV claims that their open-source library is the fastest because it is written in the native C language BoofCV claims with runtime comparison that their library is faster than OpenCV when processing higher-level algorithms [1]. However, this information is not officially tested or confirmed. Furthermore, BoofCV does not come with the amount of algorithm arsenal that OpenCV does. Poor documentation and relatively weak community support are yet another disadvantages.

After considering all above, I decided to continue with OpenCV due to the following reasons: a range of algorithms, documentation, support, and reputation.

5.4.2 Tools and Methodologies

In this part, all technology stack used for the development of the product is listed and shortly described. It comprises of programming languages, Integrated development environment (IDE), version control system and SW development methodologies.

Android Studio Android Studio is an official [9] IDE for Android application development. It is purpose-built for Android development by Google on the base of JetBrains' IntelliJ IDEA [10] and intends to accelerate development and deliver high-quality applications for any Android device.

Java Although Kotlin is Google's current preferred programming language for Android [11], Java is going to be used due to my previous experience and the fact that OpenCV does not have Kotlin interface.

[8]OpenCV, About, `https://opencv.org/about/`, accessed:31.07.2019

[9]Android website, Android Studio, `https://developer.android.com/studio/index.html`, accessed: 01.08.2019

[10]JetBrains website, IntelliJ IDEA is the base for Android Studio, the new IDE for Android developers,`https://blog.jetbrains.com/blog/2013/05/15/intellij-idea-is-the-base-for-android-studio-the-new-ide-for-android-developers/`, accessed: 01.08.2019

[11]Android website, Develop Android apps with Kotlin, `https://developer.android.com/kotlin`, accessed:01.08.2019

Python Due to its simplicity and efficiency, Python will be used in the concept phase for quick testing of different OpenCV algorithms and methods.

Git Although it is not a group project and source code is not needed to be shared with anyone during the development period, the git version control system will be employed for keeping a clean code history. Git Workflow design [12] will be utilized in order to standardize the development and make it easier for anyone to understand and continue working on the code.

Testing Test data is provided by the Supervisor. It contains a group of shooting videos from five and ten-meter distances with different camera angles. Speeds of all shoots were measured with Supido personal sports radar and documented in an excel file. This data set will be used as reference data during the development. The final product shall be compared to more accurate devices such as Doppler radar guns. Samsung Note 5 smartphone is going to be used during application development.

Agile For the development of the application, a popular approach of modern software development Agile will be utilized. As opposed to the Waterfall method, where the final product is delivered at the end of a life cycle, with Agile product is delivered constantly with small increments [13]. Although it is partly dealing with the way teams and team members collaborate with each other, the idea of incremental small deliveries of working software can be advantageous in projects like this, where a programmatic solution is not well defined beforehand and adaption to changes is required. Therefore, keeping the flexibility of switching between algorithms midway through the development when a certain approach performs weekly, will be a great asset.

In the next chapter, a range of different CV and IP methods will be tested with Python code on the macOS device and later, the selected ones will be transferred to an Android device in Java implementation. In the initial phase of development, the focus will be made on delivery of those features that meet the requirements F1 and F2 (Page 21). This decision was made due to the fact that the product may become somewhat usable by implementing just the requirements F1 and F2, but not the other way around. We will refer to that state of a product as *version one*. The remaining requirements will be fulfilled as time allows. As a result, increments in product delivery should look like following:

1. Test different methods.

[12] Atlassian website, Gitflow Workflow, `https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow`, accessed:03.08.2019

[13] Atlassian, The Agile Coach, `https://www.atlassian.com/agile`, accessed:02.08.2019

2. Build running Android application with two activities: Main and Camera activity.
3. Make camera activity to record a video.
4. Get user inputs in Main activity and pass them to Camera activity.
5. Implement ball detection and tracking.
6. Implement goal detection and tracking.
7. Calculate the speed.
8. Optimize the solution.
9. Create database infrastructure for storing shooting results.

6 Computer Vision and Image Processing Methods

As found out from theoretical solution section (Page 26), finding the positions of a goal and a ball in video frames and possessing the information about timestamp is sufficient to calculate the speed. In this chapter a range of different CV and IP methods will be tested using OpenCV's Python interface in order to find optimum algorithms and methods. It is assumed that the performance of the algorithms will be similar in Android environment with Java interface.

Main criterion used for comparison of methods is speed, since the product under development is intended for mobile devices, which are known to be limited in resources. Moreover, non-functional requirement NF2 does not allow more than ten seconds for post-processing, limiting the option to process the recording as long as it takes. The other important criterion is accuracy.

6.1 Hough Tranforms

Hough Transform (HT) is a general approach for identifying any shape that can be defined parametrically within a distribution of points [28, Sec.8.2]. Luckily, in our case the OOIs, a ball and a goalpost, are parametric shapes, circle and straight line respectively. In the following sections we are going to examine two variations of HT, Linear Hough Transform (LHT) for detecting goalpost and Circular Hough Transform (CHT) for detecting ball.

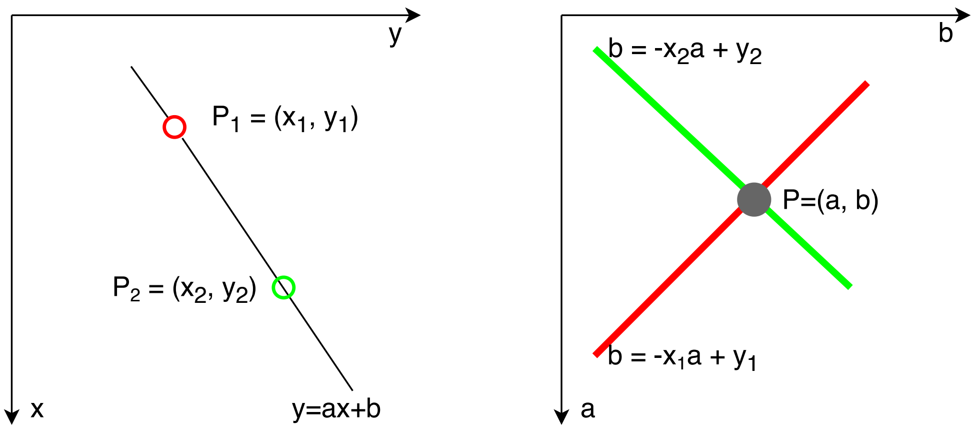
6.1.1 Line Hough Transform

As mentioned above, HT provides a means for identifying straight lines in images. It can be helpful for identifying the goal posts since in images they are vertical straight lines.

The concept behind it is point-line duality [12, p.285].The basis of HT is to find aligned points that create lines. Based on point-line duality, it is known that any point P can be represented in terms of their coordinates (x, y) or in terms of lines $y = a \cdot x + b$ passing through this point (Figure 6.1a). In the latter case, the parameters a and b are used to define the angulation of the line. In a given line there are infinite number of (x, y) tuples with different values but a and b

values of the line is fixed. Now, if we invert the plane (x, y) into a plane (a, b), each point will match to a line in (a, b) plane (Figure 6.1a). The opposite is also true, so intersection point of lines in (a, b) plane will match to the original line we depicted in (x, y) plane. LHT is based exactly on this duality. The line representation is not used in practice because for vertical lines the slope is infinite. A more practical representation is the so called *Hessian normal form* for representing the lines [28, p.165]:

$$x \cdot \cos\theta + y \cdot \sin\theta = r$$



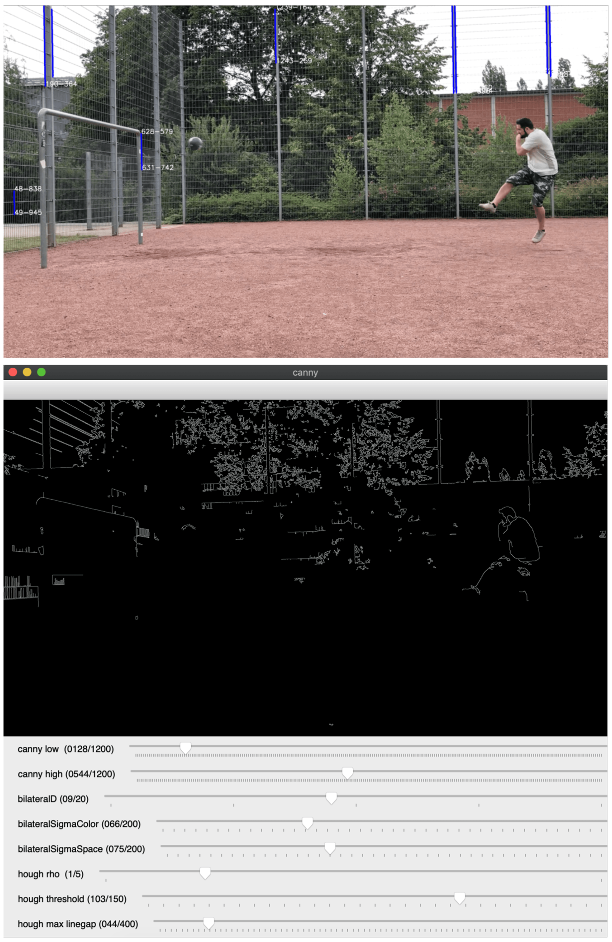
(a) Cartesian coordinate system.   (b) Feature space.

Figure 6.1: Linear Hough Transform spaces

Edge detection methods is usually applied as a preliminary processing step, outputting the information about points that represent the edges. Consequently, if edge points are aligned on the same line, the linear representation of these points on the $a - b$ plane will intersect at some point P. If there are many such edge points on the same line, there will be equally many intersections at the point P in $a - b$ plane. This point P in its turn, as mentioned above, can be translated to a specific line in $x - y$ plane.

To test LHT in work, two windows were created as shown in Figure 6.2: one displaying the detected lines on original frame, the other one displaying the output of canny operation. The panel also included a control panel with sliders to control different parameters of canny, bilateral filtering and LHT. This allowed to try different parameters in real-time and also posed a possibility to have similar panel in end product for calibration purposes.

The LHT found to be performing well in identifying the lines, depending how well the edges located in the frame. But as expected, LHT does not give the desired result right away. As displayed in Figure 6.3, it is detecting all straight lines in a given image, including vertical, horizontal, short and long lines. They were later filtered out by setting minimum length and excluding lines that span a large distance over x axis. However, this approach is not sufficient in cases like in Figure 6.2, when there are other objects on the background that produces vertical edges.



canny low (0128/1200)
canny high (0544/1200)
bilateralD (09/20)
bilateralSigmaColor (066/200)
bilateralSigmaSpace (075/200)
hough rho (1/5)
hough threshold (103/150)
hough max linegap (044/400)

Figure 6.2: Hough Lines control panel

6.1.2 Circle Hough Transform

CHT is analogues to LHT in detecting circular shapes. A circle can be described by: $(x-a)^2+(y-b)^2 = r^2$. In this equation, x, y define the center of a circle and r is the radius. Compared to LHT, three parameters (x, y, r) are needed to mathematically define the circle. As a result it requires lots of storage and computation. This was proved during testings, where CHT algorithm was taking about an order of magnitude times more processing time than LHT. Moreover, testings showed that choosing the right parameters for CHT can be tricky. As can be seen from Figure 6.4, although it works normally on sample image with ideal circles, CHT fails to detect a ball that appears the same way as in expected working condition. All these aspects, especially the performance, makes CHT a weak candidate for our application.

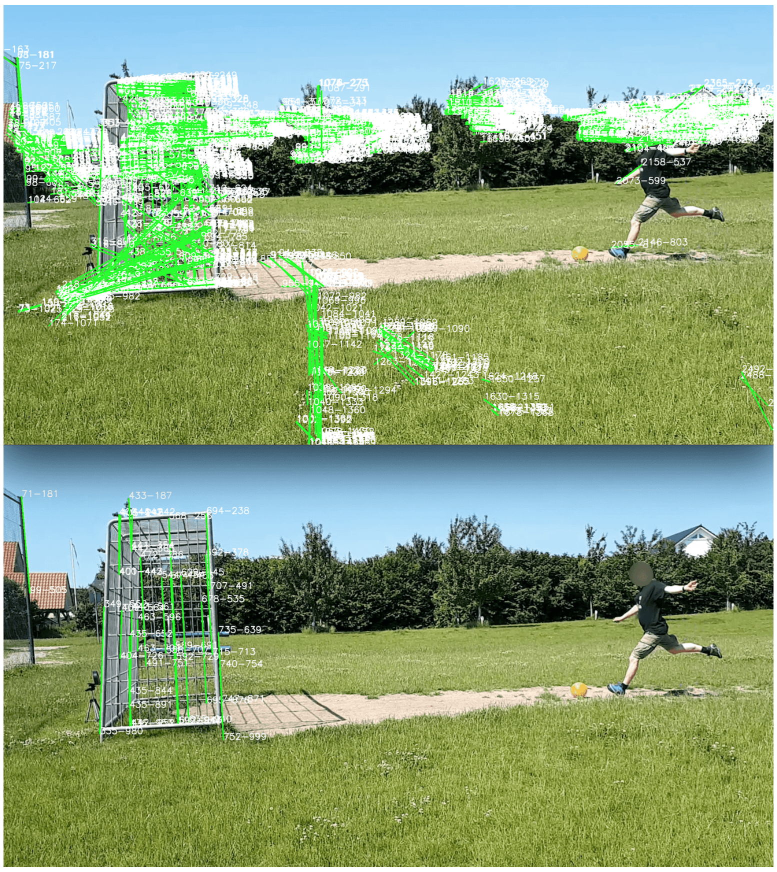


Figure 6.3: Hough Lines, before (top) vs after (bottom) filtering

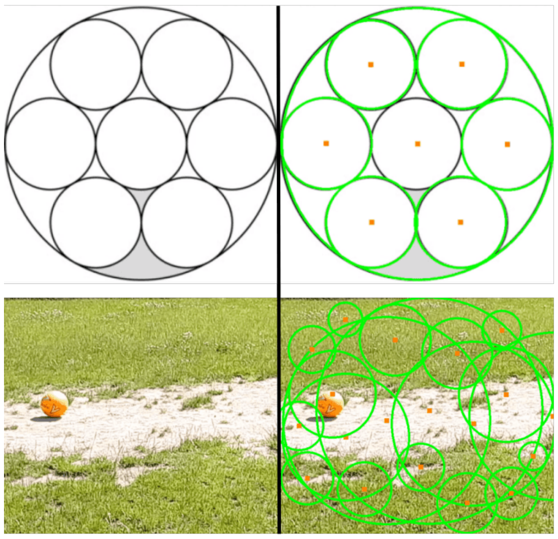
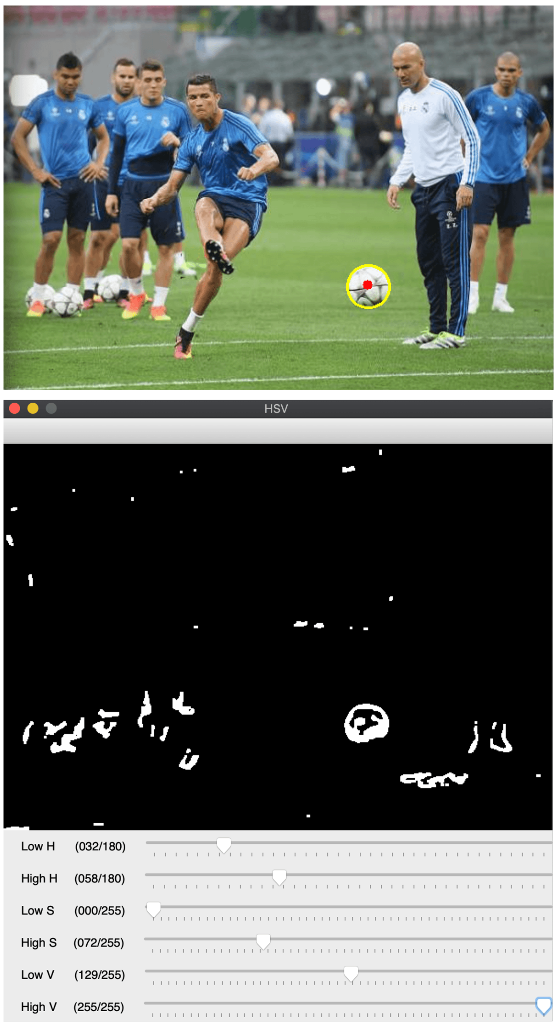


Figure 6.4: Hough Circle Transform

6.2 HSV Thresholding



Low H (032/180)
High H (058/180)
Low S (000/255)
High S (072/255)
Low V (129/255)
High V (255/255)

Figure 6.5: HSV Thresholding

To test this method, a control panel and two windows similar to the one for HT was created (Figure 6.5). Control panel allowed to manipulate low and high H, S and V values in real-time. Test images were chosen in a way that they include other objects with the similar color as the ball. To make test conditions more tricky, instead of bright ball colors that create high contrast with the background, images of balls with some patterns on them were analysed. Despite all of that, the balls were successfully identified in all test images. Moreover, complete algorithm for HSV was running faster than any other algorithm tested during this project, due to the fact that no complex preliminary or post processing being required.

Although no significant problem was encountered, this approach requires some calibration before usage, for setting proper high and low HSV range values.

6.3 Object Tracking

It is common that tracking is faster than detection. The reason is that when tracking an object that was detected in previous frames, the program possesses a lot of information about the object's actual appearance and past positions. A good algorithm will use most of the available information to identify the object in a new frame, instead of detecting it from scratch. It is also common that tracking methods may accumulate errors with every new frame and slowly drift away from the object it is tracking. To fix this problem some tracking algorithms run object tracking from time to time.

OpenCV library comes with eight different tracking algorithms. Selection of this algorithms is examined in the next sections. All algorithms require the user to manually draw a bounding box over the OOI. After that algorithms try to keep the object on track.

Identical video recordings with 30 fps and 240 fps were used for comparison. Various sizes and shapes of bounding boxes were drawn for each algorithm. Information about fps rate and status of tracker indicating whether or not OOI is under tracking was printed to the screen. Printed information is the primary indicator of how good each algorithm performs.

6.3.1 MIL

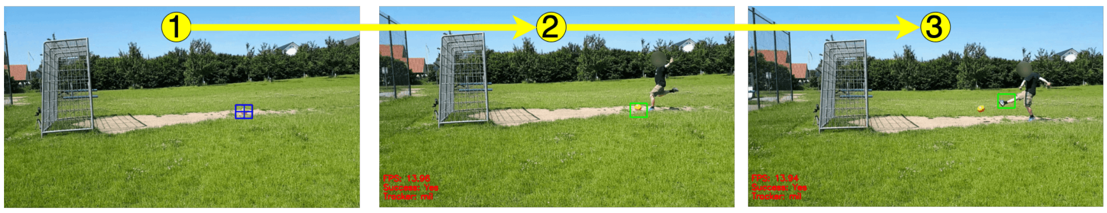


Figure 6.6: MIL tracking

This algorithm was found to be consistently failing to report the lost OOI, the condition also know as *false positive*, as can be observed from the screenshots in Figure 6.6.

6.3.2 KCF

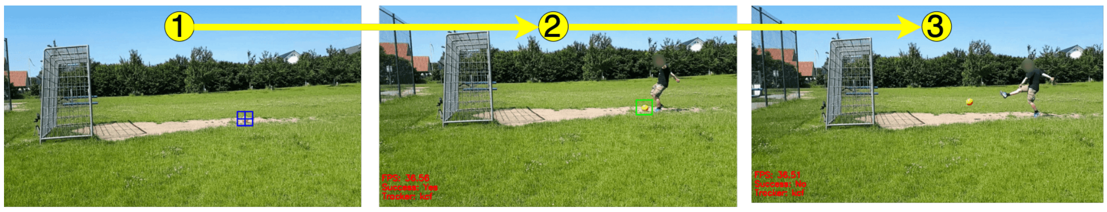


Figure 6.7: KCF tracking

Kernelized Correlation Filter tries to calculate the motion of given initial set of points by looking at the direction of change in the next frame. Once the new positions of these points are identified, the bounding box moves over. There is mathematics involved in making the search faster, which found to be true with average fps of 35, nearly 2.5 times faster than MIL at 14 fps. Moreover, it was doing better job in avoiding the false positives. As can be seen from the rightmost picture in Figure 6.7, after losing track of the ball algorithm duly reports it by not drawing a bounding box.

6.3.3 CSRT

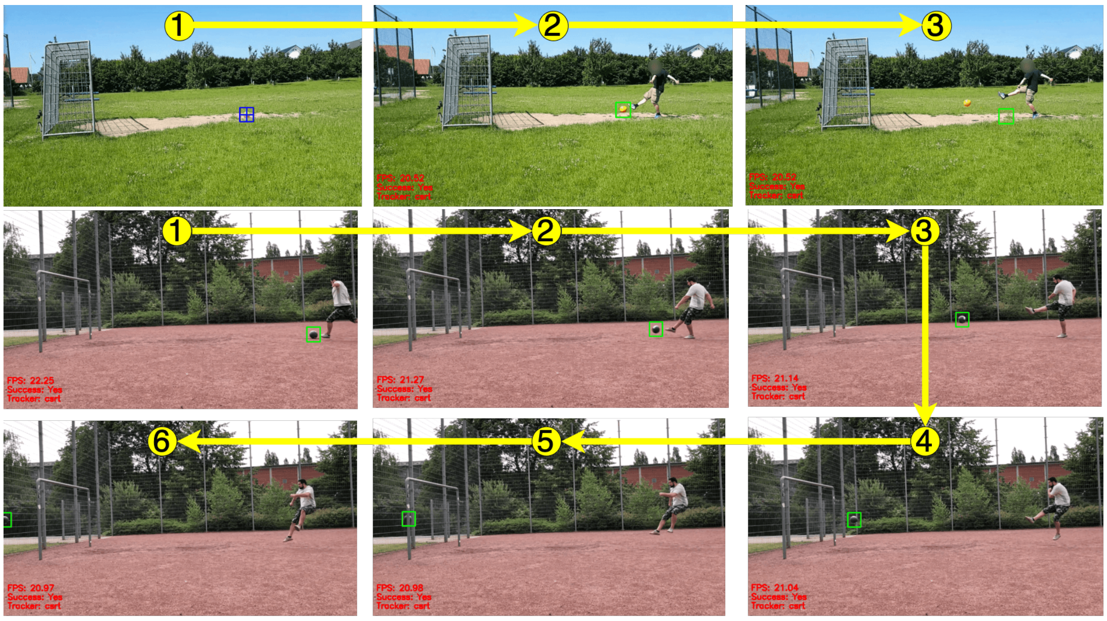


Figure 6.8: CSRT tracking failure (top) and success (bottom)

Discriminative Correlation Filter Tracker with Channel and Spatial Reliability is yet another tracking method included in OpenCV arsenal. Like MIL algorithm, CSRT was constantly showing false positives in 30 fps video. However, it demonstrated higher tracking accuracy in high fps video (Figure 6.8) compared to other two algorithms, successfully overcoming full and partial occlusions. Regarding the performance, it was in between MIL and KCF at 20 fps.

6.3.4 Object Tracking Summary

To summarize, MIL was the weakest performer due to false positives and low fps. KCF demonstrated the fastest fps throughput, but stand out with slightly lower tracking accuracy compared to CSRT, which was more accurate but still slower than KCF. Testings showed that none of these methods are good fit for tracking a fast moving object that travels a long distance (longer than its own size) between the frames. They are slow in general and perform inconsistently by heavily depending on the shape and size of a drawn bounding box.

6.4 Moving Object Detection

Moving object detection is a technique where multiple video frames are compared by various methods to determine moving objects [1]. In the following sections three most common methods will be tested, namely background subtraction, optical flow and temporal difference.

6.4.1 Background Subtraction

Background subtraction is common method for motion segmentation, but it performs best with static camera and scene. In this method a reference image is used to compare with incoming frames in order to differentiate foreground objects from the background [16][17]. Foreground objects are considered to have motion. Background objects are considered to be static and depending on the method used internally, background image can be updated if foreground object motion stops [25]. This method is easy to implement but sensitive to the changes on the background. Extracted background image is easy to be disturbed by light, weather and other environmental conditions [31]. It is better suited for use cases where background is static [5].

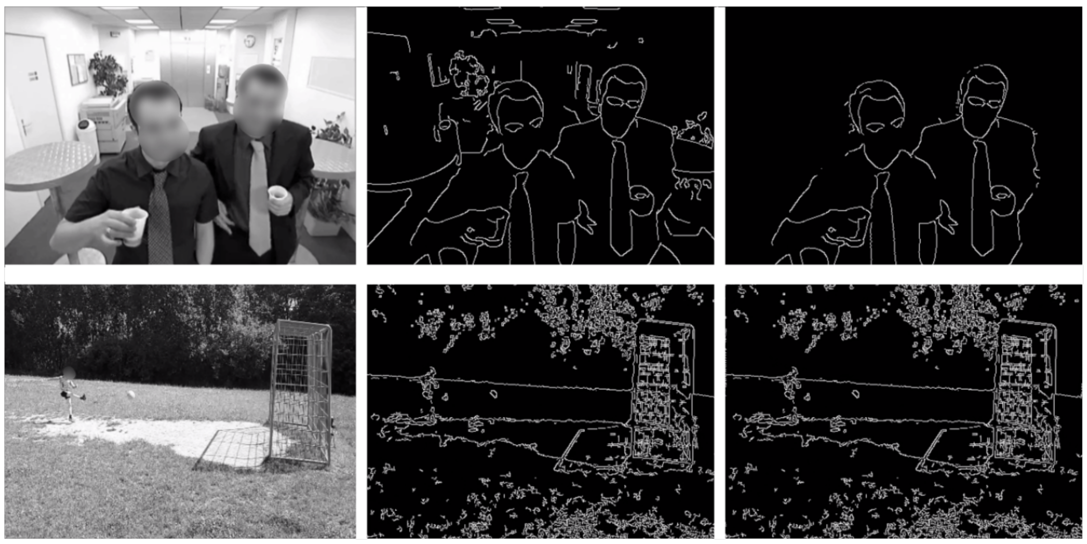


Figure 6.9: Background Subtraction implementations. Stationary (top) vs non-stationary (bottom) camera recordings

[1]Wikipedia, Moving object detection, `https://en.wikipedia.org/wiki/Moving_object_detection`, accessed: 20.07.2019

To test this method, two different video recordings were fed to the algorithm, first with static camera and the second one with non-static camera. As can be seen from Figure 6.9, the persons on the foreground were successfully detected and isolated. On the second case, however, the method did not perform well due to extensive camera movements. The second video file was one of the test videos which simulate the expected working conditions of the end product. Therefore this method is unsuitable for our purposes, unless some additional processing are done for stabilizing the the recordings.

6.4.2 Optical Flow

Optical flow method uses flow diagrams created by moving objects to recover motion at each pixel from spatio-temporal image brightness variations [14].

This method was also tested and screenshots obtained, as can be seen from Figure 6.10. Extremely low fps rates during tests demonstrated that optical flow algorithms are computationally expensive and might not be suitable for video analysis on mobile applications.

6.4.3 Temporal Difference

Temporal difference is yet another moving object detection method and is most commonly used in scenes where the camera is non-stationary [10]. Motion is detected by taking a pixel-by-pixel difference of two or three consecutive frames.

The temporal difference of three frames is implemented by finding the absolute difference between the frames $(n, n+1)$ followed by applying the same on frames $(n+1, n+2)$. Then the resulting two frames are combined into single frame by logical and-operation. Three frames version of temporal difference method helps to further reduce the effect of insignificant motions in the final result.

Compared to the previous two methods it is many times faster as it does not involve complex operations. However, this method alone does not complete the task of finding a ball. Further processing needs to be added to somehow stabilize the recording and select apart the ball from the player shooting it, who is also appearing as a moving object (Figure 6.11). Despite that, its performance makes it a promising candidate for mobile applications.

6.5 Point Feature Matching

The idea behind point feature matching is based on point correspondences between a reference and target images. It detects features (points, edges and etc.) in reference image and then

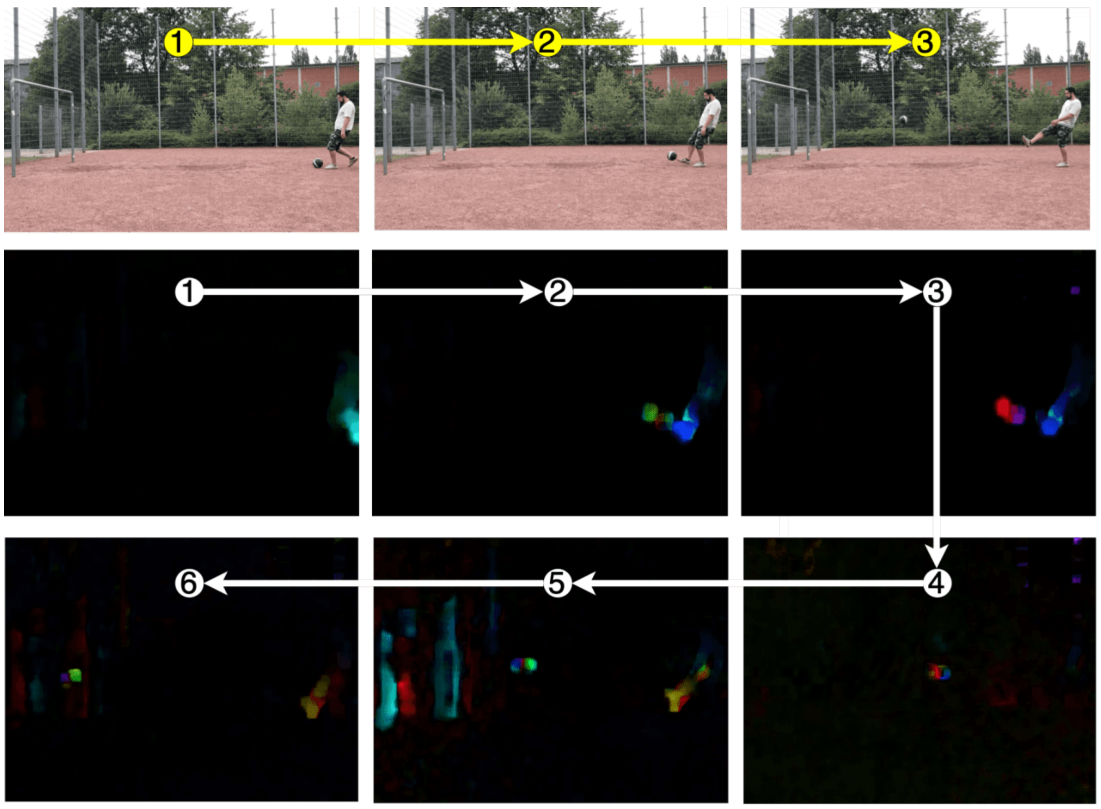


Figure 6.10: Screenshots from optical flow implementation

tries to find these points on target image. It can detect objects despite a scale change, rotation and some occlusion [2]. However, it is not expected to work well for uniformly-colored objects, which is the case with some ball types. Another problem could be the fact the fast moving ball might appear blurry making it hard to detect the feature points in target image. This method does not seem to be a good fit for our task.

6.6 Decision

After implementing and evaluating (Table 6.1) all aforementioned methods the decision was made to continue with three promising methods: LHT, HSVT and three frame temporal differencing. They are all fast and can be accurate when properly calibrated. LHT can be used

[2]MathWorks, Object Detection in a Cluttered Scene Using Point Feature Matching, `https://www.mathworks.com/help/vision/examples/object-detection-in-a-cluttered-scene-using-point-feature-matching.html`, accessed: 20.07.2019

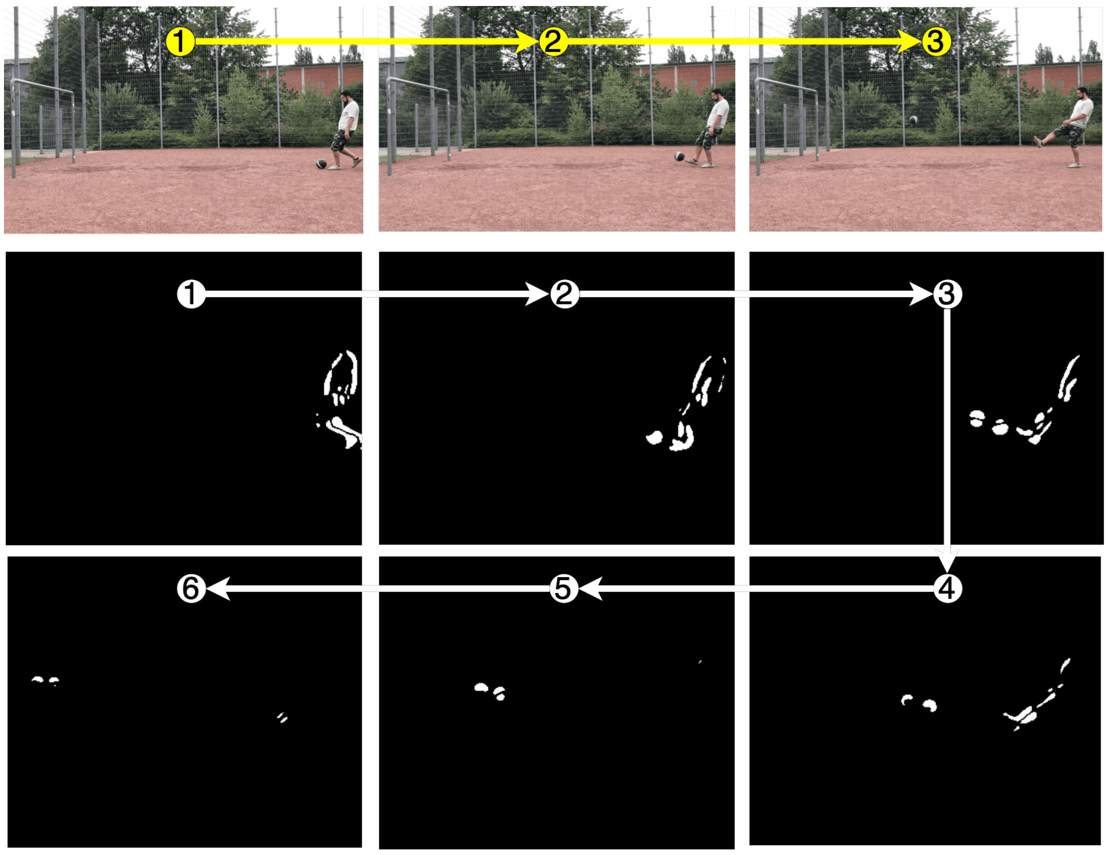


Figure 6.11: Screenshots from temporal difference implementation

for tracking the goalpost, while temporal differencing for ball tracking. HSVT can be used for both tasks. Figure 6.12 contains the screenshots of HSVT in action. The advantage of this method is that, given enough contrast of the ball with background, it is very fast, prone to camera shakes and blurs.

Although these methods might perform detection and tracking task separately, their combination will be exploited to achieve a two-fold tracking, like in the works done by other authors [9, 4], with the possibility to increase layers of tracking by even more methods, including trajectory based tracking method. Care must be taken in finding the balance between real-time and post-processing for meeting the requirements.

Table 6.1: Summary of methods

Method	Speed	Accuracy	Usage
Line Hough Transform	fast	normal	goalpost tracking
Circle Hough Transform	slow	bad	ball tracking
HSV Thresholding	fast	good	ball and goalpost tracking
MIL, KCF, CSRT	normal	normal	ball and goalpost tracking
Background subtraction	slow	bad	ball tracking
Optical flow	slow	good	ball tracking
Temporal difference	fast	good	ball tracking
Point feature matching	N/A	bad	ball and goalpost tracking

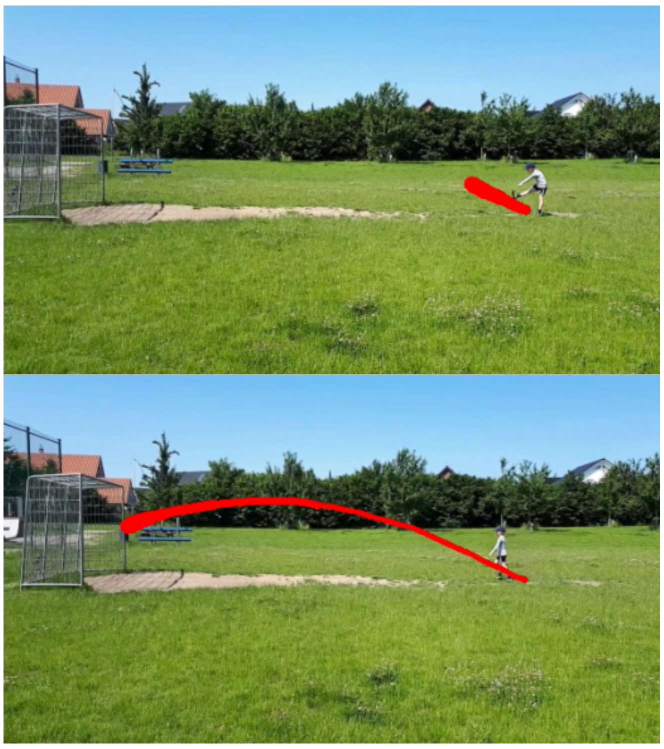


Figure 6.12: Ball tracking with HSV thresholding

7 Design

Due to time constraints, detecting and tracking the position of goalposts will not be taken into account during designs. To overcome this problem, user assistance will be required. Depending on the direction of a shooting, a fixed vertical line will be drawn on the left or right side of the screen. The operator will need to hold the camera in such a way that on the screen this vertical line positions in between two goalposts and parallel to them.

7.1 Design Alternatives

Although most of the aspects of product design are specified in the requirements and others derived during the conception phase, there are still some design details that need to be clarified. All these concerns summarized in the next paragraphs.

Measurement Modes It was decided to use a combination of different methods for tracking (Section 6.6). But for the beginning, the product will have two different modes for speed measurement: "HSV" mode and "Difference" mode. The former will exploit HSVT method and the latter TD method. LHT will be applied wherever necessary.

Processing No matter which CV method is used, their complete implementation and integration into the overall process takes many different steps like, image resizing, time stamp capturing, smoothing, color space conversions, contour finding, bounding boxes drawing, speed estimation, etc. In general, we have two options for handling all these steps:

1. Real-time. Performing all steps as soon as a video frame is available.
2. Post-processing. Performing the steps after the recording stopped.

A balance needs to be found between the two. On the one hand, there is non-functional requirement NF2 (Page 24) that limits the post-processing time at ten seconds. On the other hand complex computations in real-time may severely harm the fps rate of recording. Thus, the preference will be given to post-processing whenever there is a computationally expensive step.

Frame resizing, time stamp capturing and other simple steps can be done in real-time. At this stage of the development, every processing step and details are unknown. Therefore, the final balance between real-time and post-processing will be adjusted during the implementation phase based on actual performance. All in all, I will try to keep fps rate above 24, the rate which provides adequate visual continuity [20].

Distance There are different ways for estimating a distance between two objects directly from a video recording (e.g. one of the modes in SpeedClock application (Section 3.2.4)). Despite that decision was made to obtain the ball to goal distance from a user in the first version of the product. This decision should not harm the user experience and must bring in considerable amount of relief in terms of application development time.

Direction The information about the direction of a shoot was also decided to be gathered from a user. Like in the previous paragraph, it is believed that this approach will not damage the usability of the app. Instead, it will help to reduce the workload during development.

7.2 UML Diagrams

The Unified Modeling Language (UML) is used in the next sections for modeling some parts of the system. These diagrams will assist the development and increase efficiency.

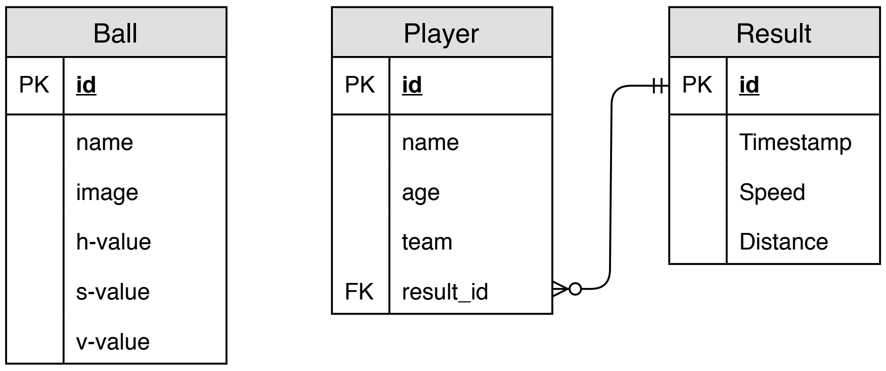


Figure 7.1: Entity-relationship model of the application

7.2.1 Entity-relationship Model

To meet the concept goals, the system needs to store some information about ball and players in the database. Android comes with built in SQL database implementation called SQLite and it supports all the relational database features. Figure 7.1 shows the structure of the database.

To fulfill the functional requirements F3, F5, F6 and F7 (Page 23) we need to save user's name, age and their shooting results at different points in time from different distances. In order to reduce data redundancy, the database structure is normalized by creating a many-to-one relationships between the tables Player and Result.

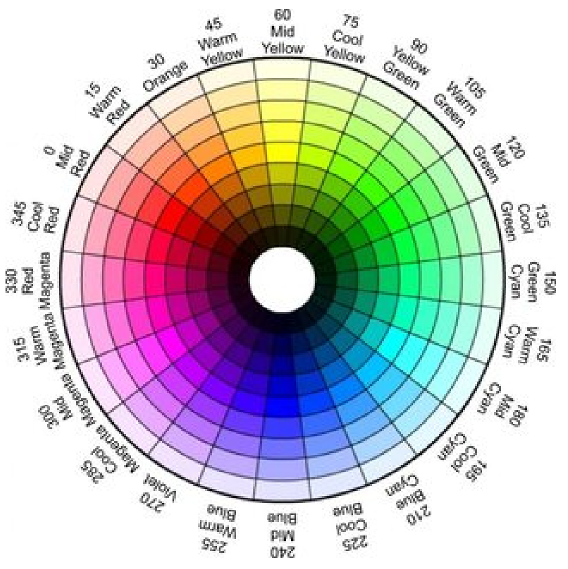


Figure 7.2: Hue color spectrum

The table Ball is necessary for "HSV" measuring mode. As can be seen from Figure 7.1, the ball's picture, along with user-defined name and HSV color values are saved. This table will be initialized with some predefined data during the first start of the application. This will be done in accordance with 24 different hue values shown in Figure 7.2. One little detail here is that, for OpenCV hue range is [0,179] [1]. To overcome this issue, the values defined in the figure will be divided into two. In addition to these colors, entries for black and white colors will also be included in the table. This will result in 26 predefined ball entries at the start. Moreover, the user will get a chance to add even more colors to the database.

7.2.2 State Diagram

As illustrated in Figure 7.3, there are five states during speed measurement activity in HSV mode. The activity starts in the Idle state and tries to detect a ball using HSVT method. If it succeeds, a bounding circle is drawn around the ball. Otherwise, user will need to calibrate HSV settings or choose one of the predefined balls explained in Section 7.2.1.

After detecting the ball, the system is ready for measurement. At this point, the user can start measuring by pressing the record (or measure) button. This takes the activity into Record

[1] OpenCV, Changing Colorspaces, `https://docs.opencv.org/master/df/d9d/tutorial_py_colorspaces.html`, accessed: 23.08.2019

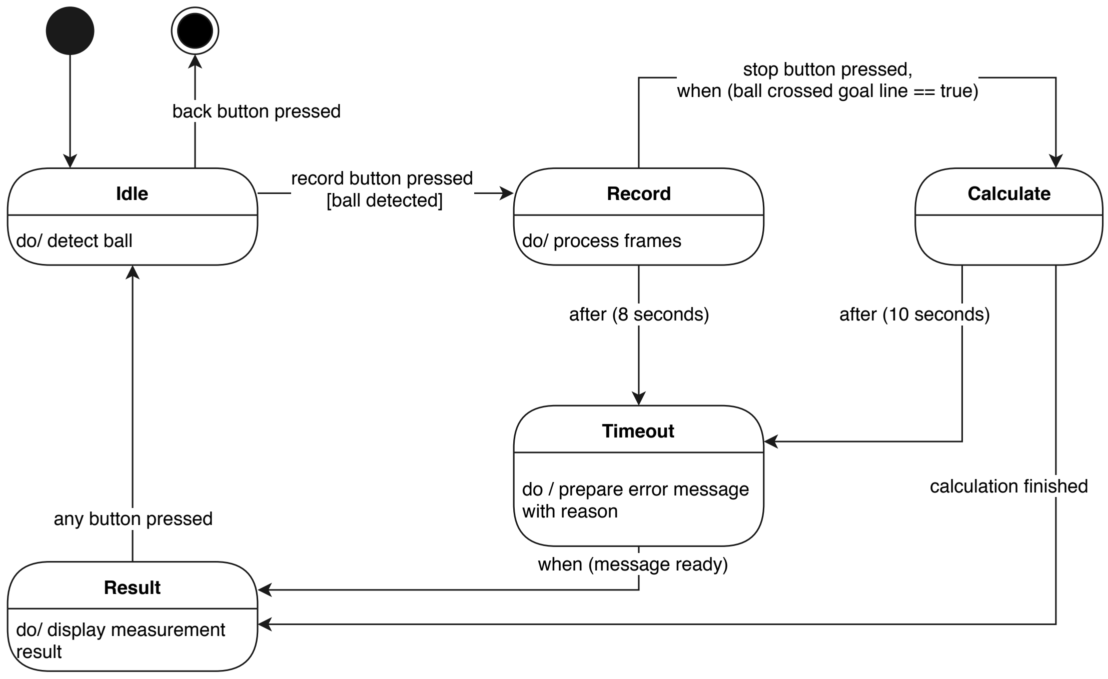


Figure 7.3: State diagram for speed measurement activity in HSV mode

state and subsequently to Calculate state. If it takes more than a specified time in any of these states, activity switches to Timeout state, where the reason for the timeout gets appended to the error message, which is later displayed during the Result state. The result state also prints the measured speed if the calculation was successful. The user goes back to Idle state after reading the message.

7.2.3 Activity Diagram

Activity diagram for use case "Measure" (Table 4.2) is shown in Figure 7.4. As illustrated, activity starts with a press of a button, as long as the ball and the goalpost are in the detected state. Otherwise, the operator cannot start the activity, because ball and goalpost positions are necessary for conducting a speed calculation. This activity can be divided into two phases: recording and calculating. Correspondingly, there are two conditions that can cause the activity to fail: timeout during the recording phase and timeout during the calculation phase.

The shift from the recording phase to the calculation phase happens in two different ways. First, during a typical flow, when the program detects that ball passed the goal line. Second, when the operator manually stops the recording. The latter is necessary to avoid unwanted failure of the activity when the ball bounces back from a goal preventing the program to detect the event. This might lead to a timeout of the recording phase which is defined as eight seconds due to FM7 (Page 29).

The duration of the calculation phase is limited to ten seconds due to NF2 (Table 4.4). During this time the result of calculations is made available to the system and activity ends.

7.3 Graphical Design

7.3.1 UI/UX

To create color schemes, Google's Material Design Color Tool [2] is used. This tool automatically derives light and dark versions of chosen primary and secondary colors. The resulting colors is easily exported in XML format (Listing 7.1) for the usage in Android application.

```
    <color name="primaryColor">#7e57c2</color>
    <color name="primaryLightColor">#b085f5</color>
    <color name="primaryDarkColor">#4d2c91</color>
    <color name="secondaryColor">#fdd835</color>
    <color name="secondaryLightColor">#ffff6b</color>
    <color name="secondaryDarkColor">#c6a700</color>
    <color name="primaryTextColor">#ffffff</color>
    <color name="secondaryTextColor">#000000</color>
```

Listing 7.1: Primary and secondary color codes

Background image of the Home screen is taken from a stock photography sharing website Unsplash. Their license gives a right to copy, modify, distribute and use the photos for free, including commercial purposes, without asking permission from or providing attribution to the photographer or Unsplash [3].

To improve user experience, simple navigation is tried to be achieved by minimizing the number of screens and exploiting pop-ups. Moreover, distance selection is implemented in a *slider* fashion, which lays intuitively in between a goal and ball icons, indicating what value is being changed, and removes the necessity of typing the numbers manually, which would have resulted in more than three screen touches compared to a single touch with a slider.

7.3.2 Wireframe

Blueprint of the application layout is illustrated in Figure 7.5. There are three main screens: Home, Statistics and Radar, in addition to pop-up screens.

[2]Material Design, Color Tool, `https://material.io/resources/color`, accessed: 20.08.2019
[3]Unsplash, License, `https://unsplash.com/license`, accessed: 23.08.2019

The home screen is the starting point of the application. Users can press the Statistics button for search and sort activities. User sets the ball to goal distance with a slider. Users can also select the type of ball from the "carousel" component of this screen. The component will contain predefined balls built-in in the application representing 26 different colors as explained in Page 50. The new ball can be added by pressing the Save button from the Settings pop-up menu. Screenshot of the ball will be taken and saved as long as there is a single circular shape on the screen after applying HSV thresholding. To measure the speed, a user has to go to the Radar screen. After the successful auto-detection of the ball, a record button is enabled for pressing. Once pressed, recording starts and stops after a specified time or by user. As soon as the recording is over, Processing pop-up appears and depending on result leads to Success or Error pop-ups. Users can calibrate the tracking by manipulating threshold values via settings icon on the Radar screen.

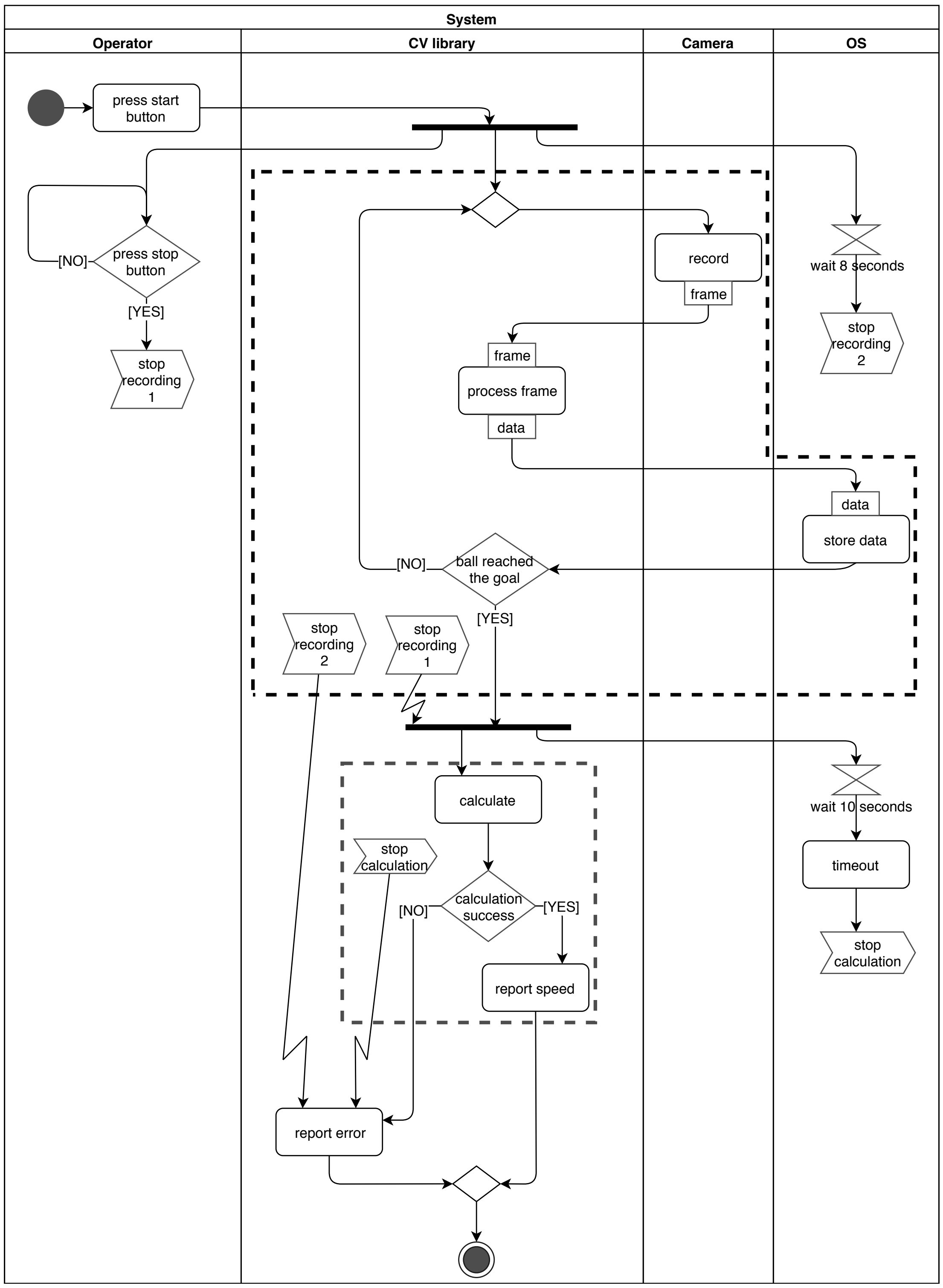
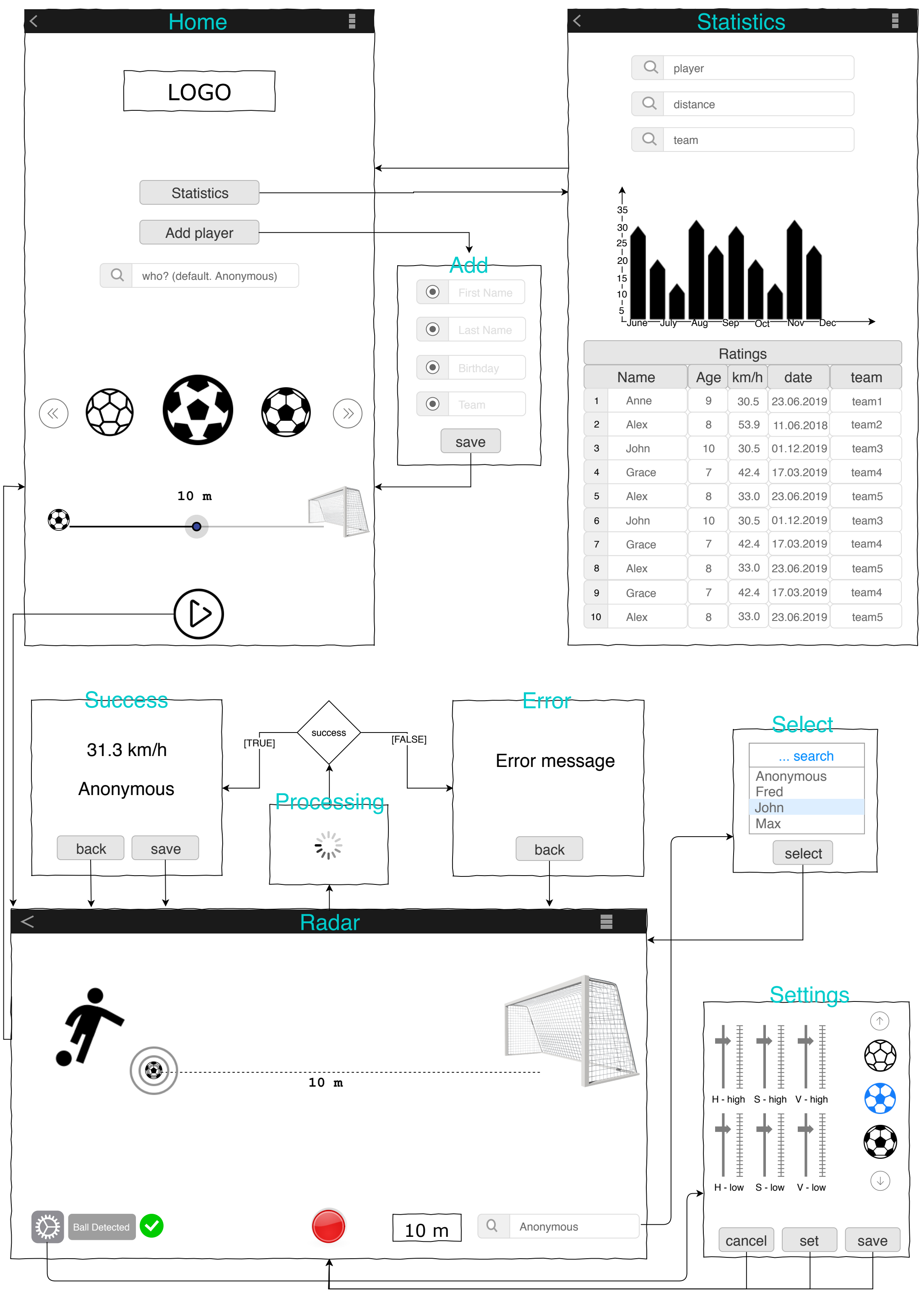


Figure 7.4: Activity diagram for the use case "Measure speed"



Ratings

	Name	Age	km/h	date	team
1	Anne	9	30.5	23.06.2019	team1
2	Alex	8	53.9	11.06.2018	team2
3	John	10	30.5	01.12.2019	team3
4	Grace	7	42.4	17.03.2019	team4
5	Alex	8	33.0	23.06.2019	team5
6	John	10	30.5	01.12.2019	team3
7	Grace	7	42.4	17.03.2019	team4
8	Alex	8	33.0	23.06.2019	team5
9	Grace	7	42.4	17.03.2019	team4
10	Alex	8	33.0	23.06.2019	team5

Figure 7.5: Wireframe of the application

8 Implementation

8.1 Setup and Configuration

Development of the application carried out in Android Studio IDE. To enable OpenCV, the latest[1] OpenCV Android library was downloaded and imported into the project. Rearview camera[2] permission is requested in the Android manifest file.

There is also an option to use the OpenCV library in the original C++ language by exploiting Android NDK (Native Development Kit) toolset. But many reviews from OpenCV and Stackoverflow websites hint that calls from Java to NDK can be really expensive in terms of performance.

8.2 Coding

Preliminary processing To start the measurements, information about the ball-to-goal distance and direction of the shoot is required. These are provided by the user at the start. When the user presses the record button, video frames start to be delivered to a Java function. These frames are smoothed using Gaussian Blur function. After that, the frame is ready for processing by HSVT and TD methods.

Ball detection algorithm

1. HSV Thresholding. The snapshot of the code can be seen in Listing 8.1. The image supplied by the Android camera is available in grayscale and RGB space. The color space is converted to HSV space and OpenCV's *inRange* function is used to threshlold the image with given upper and lower HSV values. This function returns a binary mask, where white pixels represent pixels that fall into the upper and lower limit range and black pixels do not. The mask might include some noise. We eliminate the unwanted small white pixel areas by erosion followed with dilation, the operation also known as

[1]At the time of writing, the latest available version was 3.4.3.
[2]Necessary for letting the user to easily interact with phone display while recording.

opening. Next, *findContours* method is used for finding the image regions having the same intensity. In the best case, the resulting contour is our OOI. Otherwise, if there are other similarly colored objects in the image then further processing needs to be done to filter them out.

```java
@Override
public Mat onCameraFrame(CameraBridgeViewBase.
    ↪CvCameraViewFrame inputFrame) {
mOriginal = inputFrame.rgba();
Imgproc.GaussianBlur(mOriginal, mOriginal,new Size(15,15),
    ↪0);
Imgproc.cvtColor(mOriginal, mHsv, Imgproc.COLOR_BGR2HSV);
Core.inRange(mHsv, scalarLow, scalarHigh, mMask);
Mat strcuturingElementErode = Imgproc.getStructuringElement(
    ↪Imgproc.MORPH_RECT, new  Size(2*erosion_size + 1, 2*
    ↪erosion_size+1));
Mat strcuturingElementDilate = Imgproc.getStructuringElement
    ↪(Imgproc.MORPH_RECT, new  Size(2*dilation_size + 1, 2*
    ↪dilation_size+1));
Imgproc.erode(mMask, mErode, strcuturingElementErode);
Imgproc.dilate(mErode, mDilate, strcuturingElementDilate);

List<MatOfPoint> contours = new ArrayList<>();
Imgproc.findContours(mDilate, contours, mat4, Imgproc.
    ↪RETR_EXTERNAL, Imgproc.CHAIN_APPROX_SIMPLE);
//... code for drawing bounding box
return mOriginal;
}
```

Listing 8.1: Ball detection with HSV Thresholding

2. Temporal Difference. The Listing 8.2 demonstrates the implementation of this method. The algorithm can be started after the first three frames of the recording. Once available, TD for three frames is calculated by measuring the absolute difference between frames $n+1$ and $n+2$, followed by applying the same method to frames n and $n+1$. The two frames obtained from absolute difference operations are combined into a single frame using logical and-operation. The result is a binary image, where white pixels indicate objects that had motion between frames.

```java
@Override
```

```
public Mat onCameraFrame(CameraBridgeViewBase.CvCameraViewFrame
    ↪inputFrame) {
    mOriginal = inputFrame.rgba();
    Imgproc.cvtColor(mOriginal, mOriginal, Imgproc.
        ↪COLOR_BGR2GRAY);
    Imgproc.GaussianBlur(mOriginal, mOriginal, new Size(15, 15),
        ↪ 0);
    if (counter == 0) {
        frame1 = mOriginal;
        counter++;
        return frame1;}
    if (counter == 1) {
        frame2 = mOriginal;
        counter++;
        return frame2;}
    frame3 = mOriginal;
    Core.absdiff(frame2, frame3, delta1);
    Imgproc.threshold(delta1, delta1, 5, 255, Imgproc.
        ↪THRESH_BINARY);
    Core.absdiff(frame1, frame2, delta2);
    Imgproc.threshold(delta2, delta2, 5, 255, Imgproc.
        ↪THRESH_BINARY);
    Core.bitwise_and(delta1, delta2, output);
    final List<MatOfPoint> contours = new ArrayList<>();
    Imgproc.findContours(output, contours, delta2, Imgproc.
        ↪RETR_EXTERNAL, Imgproc.CHAIN_APPROX_SIMPLE);
    //... code for drawing bounding box
    return output;
```

Listing 8.2: Ball detection with three frame temporal differencing

Post processing Previous stages involved detection and tracking of the ball. This phase would include actual calculations of the speed. Unfortunately, time constraints did not allow to accomplish this task. Having the coordinate information of the ball from previous stages and corresponding timestamp at each position, speed calculations can be done based on the theoretical solution proposed in Section 5.2.

9 Discussion and Conclusion

In this chapter, the reader is presented a general overview of what is being achieved, which problems occurred and how they tackled. It follows with a discussion of possible future work that can be developed on our work.

9.1 Results

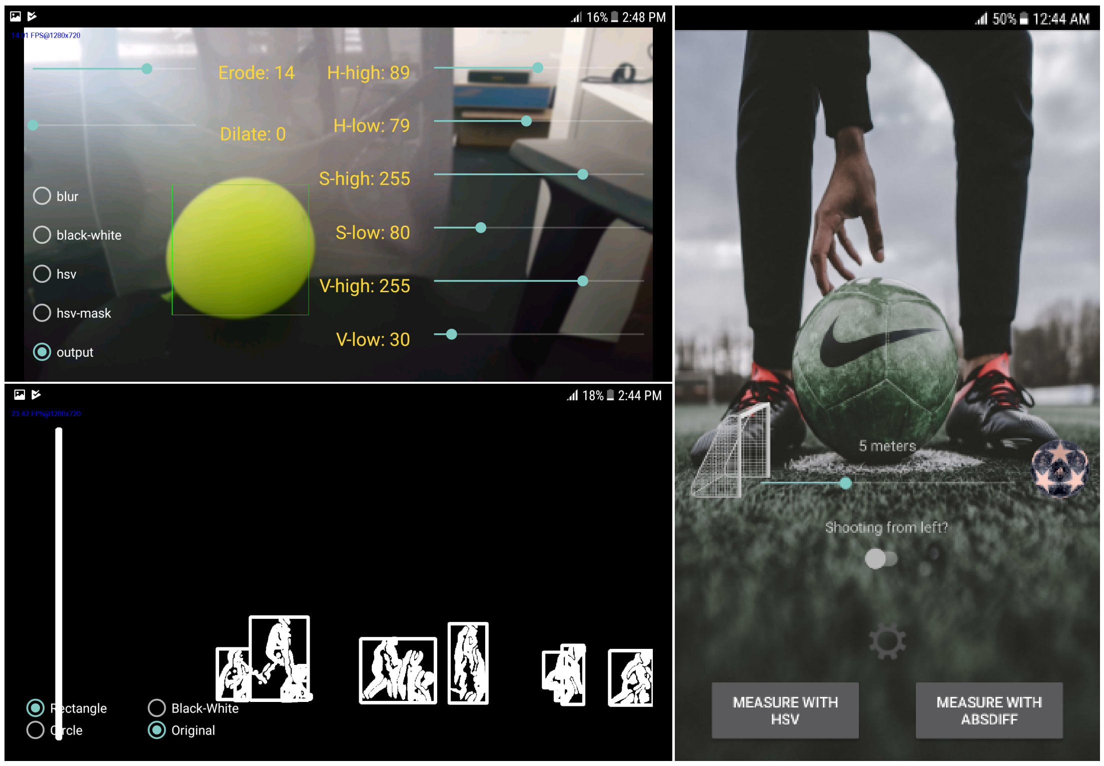
In current state, the application has two measurement modes in the form of HSVT and TD implementations. The end result of development can be seen from the actual screenshots presented in Figure 9.1. One of the screens shows how walking pedestrians were detected and bounding boxes drawn around them in TD mode. On the other one, a green ball was detected using HSVT. This screen lets the user to manipulate , erosion and dilation parameters. Users can also switch between different views to see the results of certain steps of the whole algorithm. The last screenshot from the figure presents the Main or Home screen of the application.

Although a lot of work has been done on all areas of the development of this project, none of the functional requirements were fully met. The main problems encountered were time factor and unfamiliarity with any of the crucial development environments (Android, OpenCV). Since another big portion of the project time was spent on its documentation in the form of this paper, three months do not seem to be a sufficient period for complete development of such a project with little previous knowledge.

To speed up the development of working software, some compromises have been made. Instead of detecting a goalpost automatically, a vertical line is drawn on the left or right side of the screen depending on the shooting direction. Thus, during the recordings, the camera needs to be positioned in such a way that the vertical line appears in between and parallel to two goalposts. In future work, the vertical line approach should be replaced with automatic goalpost detection using LHT or other suitable methods.

Another compromise was made on a two-fold ball tracking approach that was described in Page 45. Instead, HSVT and TD methods were implemented separately. However, it can be a good starting point for future work. The tracking results of both algorithms can be merged for more accurate results.

Due to the fact that the final product is not in a working state, accuracy tests were not conducted. After developing the first version of the product, it should be tested against commercial Doppler guns, preferably from different manufacturers. All devices should measure the speed of the same shot and the results should then be compared. As many shots as possible with different speeds should be measured. Videos should be recorded from different angles at different lightening conditions. After that, an objective conclusion can be made on the accuracy of the solution.



Erode: 14
Dilate: 0
H-high: 89
H-low: 79
S-high: 255
S-low: 80
V-high: 255
V-low: 30
blur [ ] black-white [ ] hsv [ ] hsv-mask [✓ output]
[✓ Rectangle] Circle
Black-White [✓ Original]
5 meters
Shooting from left?

Figure 9.1: Screenshots from Android application

9.2 Future Work

As mentioned in Section 9.1, future work can start with the implementation of two-fold tracking by merging the results of existing tracking modes. Next, the focus should be made on speed calculation, after which automatic goalpost detection and database features can be completed.

Considering that the different CV methods perform better at different conditions, one can go a step further and give a user the possibility to choose between different modes. Another improvement should be the exploitation of phone sensors, namely gyroscope, and accelerometer. A gyroscope can be used for determining the orientation of the camera automatically, thus removing the assumption we made back in Section 5.1 about orientation. An accelerometer, on the other, can be employed to help eliminate the assumption about stability.

One of the most important considerations in such CV applications is the quality of the camera. Because it is the major sensor or an "eye" of the system. High picture quality can drastically improve object detection. High fps rates, on the other hand, may contribute to the accuracy of the system. Modern smartphones come with a so-called "slow-motion" mode, which allows capturing videos at a whopping rate of 240 fps [1]. These devices are also increasingly powerful every year. My recommendation for future work is to remove the non-functional requirements NF6 and NF7, that asks to support older devices. Taking these requirements out can bring more possibilities in terms of freedom in the selection of a device. Continuing this work on one of the latest smartphones one can develop a very advanced measuring products that can also have huge commercial value.

The idea behind this product is very interesting and can be extended to include even more features such as tips to improve the shooting speed, motivating daily videos, sharing the results on online platforms, emailing the results to particular people (e.g. parents of the kids), etc.

9.3 Conclusion

Although the end product is incomplete, the findings of this paper show that speed measurement tasks can be accomplished with mobile phones using a single camera. The existence of commercial products such as Athla Velocity (Page 17) supports this statement. However, as a user, one needs to be aware of possible inaccuracies. But, on the other hand, similar inaccuracies are also found in other products in the market. Therefore, it can be concluded that a smartphone application can replace at least some of the existing commercial equipment for measuring the speed of a soccer ball.

[1] Apple, iPhone X - Technical Specifications, `https://support.apple.com/kb/sp770?locale=en_US`, accessed: 05.09.2019

Bibliography

[1] Peter Abeles. *Performance of OpenCV vs BoofCV: March 2019*. https://boofcv.org/index.php?title=Performance:OpenCV:BoofCV. (accessed: 16.08.2019).

[2] *About*. Hawk-Eye. URL: https://www.hawkeyeinnovations.com/about. (accessed: 16.08.2019).

[3] Peggy Adamson and Jackie Nicholas. *Introduction to Trigonometric Functions*. University of Sydney, 1998.

[4] Pallavi et al. "Ball detection from broadcast soccer videos using static and dynamic features". In: *Journal of Visual Communication and Image Representation, 19(7), pp.426–436* (2008).

[5] Er. Monica Goyal Amandeep. "Review: Moving Object Detection Techniques". In: *IJC-SMC, Vol. 4, Issue. 9* (2015).

[6] *Arduino Uno technical specifications*. Rev. 3. Arduino. URL: https://store.arduino.cc/arduino-uno-rev3. (accessed: 10.08.2019).

[7] The Football Association. *The Football Association: The FA Guide To Pitch and Goal Post Dimensions*. Brochure. Wembley Stadium, PO Box 1966, London SW1P 9EQ, 2012. URL: https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=3&cad=rja&uact=8&ved=2ahUKEwiFrJLlqYfkAhWHyKQKHd37C_UQFjACegQIABAC&url=http%3A%2F%2Fwww.thefa.com%2F-%2Fmedia%2Fcfa%2Flancashirefa%2Ffiles%2Fleagues-and-clubs%2Frules-and-regulations%2Fthe-fa-guide-to-pitch-and-goalpost-dimensions.ashx%3Fla%3Den&usg=AOvVaw2b1yWVupOZafW79CmcSDod. (accessed: 16.08.2019).

[8] The International Football Association Board. *The International Football Association Board: Laws of the game, 2018/2019*. Brochure. Münstergasse 9, 8001 Zurich, Switzerland, 2017. URL: https://img.fifa.com/image/upload/khhloe2xoigyna8juxw3.pdf. (accessed: 16.08.2019).

[9] Bodhisattwa Chakraborty and Sukadev Meher. “A real-time trajectory-based ball detection-and-tracking framework for basketball video”. In: *Journal of Optics 42.2, pp. 156–170.* (2013).

[10] Chun Yu Chen and Ming Yue Zhao. “Video segmentation algorithm based on improved kirsch edge operator and three-frame difference”. In: *Advanced Materials Research. Vol. 981. Trans Tech Publ. pp. 335–339* (2014).

[11] Christopher M. Brown Dana H. Ballard. *Computer Vision.* Prentice Hall, 1982.

[12] Roy Davies. *Computer and Machine Vision: Theory, Algorithms, Practicalities.* Academic Press, 2012.

[13] Alessandro Verri Emanuele Trucco. *Introductory Techniques for 3-D Computer Vision.* Prentice Hall, 1998.

[14] Zizilia Zamudio Beltrán Eric Hernández Castillo and Juan Manuel Ibarra Zannatha. “Soccer ball speed estimation using optical flow for humanoid soc- cer player”. In: *Electronics, Robotics and Automotive Mechanics Conference (CERMA), pp. 178–183* (2011).

[15] M.J. Fry and J.W. Ohlmann. “Introduction to the special issue on analytics in sports, Part 1: General Sports Applications”. In: *Interfaces, 42, pp.105–108* (2012).

[16] R.P Patil Mahamuni P.D and H.S.Thakar. “Moving Object Detection Using Background Subtraction Algorithm using simulink”. In: *IJERT Volume 3, Issue 6* (2014).

[17] Othman O. Khalifa Mahmoud Abdulwahab Alawi. “Performance Comparison of Background Estimation Algorithms for Detecting Moving Vehicle”. In: *World Applied Sciences Journal 21 (Mathematical Applications in Engineering): pp. 109-114* (2013).

[18] Masanori Kano et al. Masaki Takahashi Kensuke Ikeya. “Robust Volleyball Tracking System using Multi-View Camerasv”. In: *23rd International Conference on Pattern Recognition (ICPR)* (2016).

[19] David Murdock. *Worked Examples from Introductory Physics Vol. I: Basic Mechanics.* Tennessee Tech University, 2005.

[20] Mark-Paul Meyer Paul Read. *Restoration of motion picture film.* Caleidoscope, 2000, p. 24.

[21] *Personal Sports Radar Instruction Manual.* Supido. URL: `https://www.e-sportshop.cz/data/navody/028462eng.pdf`. (accessed: 16.07.2019).

[22] Florian Ion Petrescu. *A new Doppler Effect.* Books on Demand GmbH, 2012.

[23] *Raspberry Pi 3 technical specifications.* Raspberry Pi. URL: `https://www.raspberrypi.org/magpi/raspberry-pi-3-specs-benchmarks/`. (accessed: 10.08.2019).

[24] P. Shimpi et al. S. Kolkur D. Kalbande. “Human Skin Detection Using RGB, HSV and YCbCr Color Models”. In: *Advances in Intelligent Systems Research. Vol. 137, pp. 324-332* (2017).

[25] Khalid Saeed Soharab Hossain Shaikh and Nabendu Chaki. “Moving Object Detection Using Background Subtraction”. In: *Springer* (2014).

[26] John C. Sparks. *The Pythagorean Theorem. Crown Jewel of Mathematics.* AuthorHouse, 2008.

[27] Richard Szeliski. *Computer Vision: Algorithms and Applications.* 2010.

[28] Mark J. Burge Wilhelm Burger. *Digital Image Processing. An Algorithmic Introduction Using Java.* Springer, 2016.

[29] Math Works. *Image Types.* URL: `https://www.mathworks.com/help/matlab/creating_plots/image-types.html`. (accessed: 17.08.2019).

[30] Chern-Horng Simh et al. Xinguo Yu Changsheng Xu. “A Trajectory-based ball detection and tracking with applications to semantic analysis of broadcast soccer video”. In: *Processing, pp.1049–1052* (2004).

[31] Jiamin Zheng Xuegang Hu. *An Improved Moving Object Detection Algorithm Based on Gaussian Mixture Models.* 2016.

[32] F. Yan, W. Christmas, and J. Kittler. “A Tennis Ball Tracking Algorithm for Automatic Annotation of Tennis Match”. In: *Proceedings of the British Machine Vision Conference, pp.67.1-67.10* (2005).

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 23. September 2019 Nizami Zamanov