HAW HAMBURG

BACHELORTHESIS
Jan Hendrik Wüpper

# Entwicklung von Methoden der Bildverarbeitung und einer prototypischen Android-Anwendung zur Messung von Ballgeschwindigkeiten verschiedener Sportarten

FAKULTÄT TECHNIK UND INFORMATIK
Department Informations- und Elektrotechnik

Faculty of Computer Science and Engineering
Department of Information and Electrical Engineering

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Jan Hendrik Wüpper

# Entwicklung von Methoden der Bildverarbeitung und einer prototypischen Android-Anwendung zur Messung von Ballgeschwindigkeiten verschiedener Sportarten

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Elektro- und Informationstechnik*
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Marc Hensel
Zweitgutachter: Prof. Dr. Heike Neumann

Eingereicht am: 12. März 2021

**Jan Hendrik Wüpper**

**Thema der Arbeit**

Entwicklung von Methoden der Bildverarbeitung und
einer prototypischen Android-Anwendung zur Messung von Ballgeschwindigkeiten verschiedener Sportarten

**Stichworte**

Bildverarbeitung, Computer-Vision, Android, Geschwindigkeitsmessung, Sport, Ballgeschwindigkeit

**Kurzzusammenfassung**

Diese Arbeit beschäftigt sich mit der konzeptionellen Erstellung einer App, die durch Bildverarbeitung die Geschwindigkeiten von Bällen verschiedener Sportarten messen kann. Ein besonderes Augenmerk liegt dabei beim Fußball. Die Arbeit umfasst die Entwicklung performanter und präziser Algorithmen der Bildverarbeitung, sowie die Entwicklung einer Prototyp-App.

**Jan Hendrik Wüpper**

**Title of Thesis**

Development of image processing methods and an prototype android-app for the speed measurements of balls in different sports

**Keywords**

image processing, computer-vision, android, speed measurement, sports, ball speed

**Abstract**

This thesis deals with the conceptual creation of an app that can measure the speeds of balls of different sports through image processing. A special focus is on soccer. The work includes the development of performant and precise algorithms of image processing, as well as the development of a prototype app.

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Listings

# 1 Einleitung

Immer rechenstärkere mobile Endgeräte ermöglichen es immer komplexer werdende Algorithmen auch für den alltäglichen Gebrauch zu nutzen. Ein Beispiel hierfür ist die Bildverarbeitung. Moderne Kamera-Apps erfassen Gesichter in einem Bruchteil einer Sekunde oder können eine große Menge unterschiedlicher Szenen[1] unterscheiden, um danach die Kameraeinstellungen anzupassen. Sozial-Media-Apps bieten die Möglichkeit, anhand verschiedenster Filter, das eigene Selfie noch interessanter für die Follower zu gestalten. Doch auch ganz praktische Anwendungsmöglichkeiten ergeben sich daraus. Eben eine solche Anwendung ist Gegenstand dieser Arbeit.

## 1.1 Motivation

In vielen Ballsportarten ist es von Vorteil ein Schuss- oder Wurfkrafttraining durchzuführen, um die eigenen Fähigkeiten gezielt weiterzuentwickeln. Dies gestaltet sich jedoch als schwierig, oder zumindest wenig zielführend, wenn der Trainingsfortschritt dabei nicht gemessen werden kann. Es ist daher von Vorteil, wenn es beim Training die Möglichkeit gibt die Ballgeschwindigkeit messen zu können. Um dies zu erreichen, gibt es bereits unterschiedliche Ansätze. Diese entpuppen sich jedoch als äußerst kostspielig oder wenig genau. Dabei sind die Möglichkeiten für das Umsetzen einer genauen Softwareanwendung für mobile Endgeräte gegeben. Durch Bildverarbeitungsalgorithmen, die auch auf mobilen Geräten implementierbar sind, ist es beispielsweise möglich Gegenstände in einem Bild zu detektieren oder Bewegungen zu erkennen. Eine derartige Anwendung würde keine Mehrkosten für einen potenziellen Nutzer dieser App bedeuten, da keine zusätzliche Hardware gebraucht wird und die meisten dieser Nutzer über ein Smartphone verfügen. Die Inspiration für diese Arbeit ist das in Kapitel 3.1 vorgestellte Radar-Messgerät. Dieses

[1] Gemeint sind verschiedene Aufnahmesituationen, nach denen die Kameraeinstellungen vorgenommen werden können. Beispiele sind: Sonnenuntergang, Essen, Landschaft oder Porträt.

ermöglicht es beispielsweise beim Fußball, ein Schusskrafttraining durchzuführen. Allerdings ist das Radar-Messgerät aber relativ kostspielig und dadurch für kleine Vereine und Privatpersonen nicht immer einsetzbar. Aus diesem Grund war es die Idee von Professor Dr. Marc Hensel eine App zu entwickeln, die es möglich macht diese Aufgabe zu erledigen, ohne dabei weitere Kosten zu erzeugen.

## 1.2 Thema und Ziel der Arbeit

Die eben angesprochenen Möglichkeiten werden in dieser Arbeit genutzt, um den Grundbaustein für eine mobile Anwendung zu legen, die die Geschwindigkeiten von Bällen verschiedener Sportarten mit hinreichender Genauigkeit messen kann. Dies geschieht, indem die Ballbewegung mit der Smartphone-Kamera aufgezeichnet und automatisch mittels Bildverarbeitung ausgewertet wird. Entstehen soll das Konzept einer App, die in verschiedenen Sportarten zum Training genutzt werden kann. Für die Umsetzung dieses Vorhabens sind mehrere Teilaspekte zu erarbeiten. Zum einen müssen die Algorithmen zur Erkennung des Balles entworfen und implementiert werden. Diese sollen die Ballposition (relativ zur Kamera, da diese als Messinstrument verwendet wird) bestimmen und daraus die Geschwindigkeit ermitteln. Zwar sollen die Algorithmen auf jeden Ball anwendbar sein, ein besonderer Fokus wird dabei aber auf den Anwendungsbereich Fußball gelegt. Dieser Teil, die Entwicklung dieser Algorithmen, stellt den Hauptteil dieser Arbeit dar. Zum anderen werden die Grundlagen für eine Implementierung dieser Algorithmen in einer App geschaffen. Dazu werden verschiedene Möglichkeiten zum Laden der Bilder von der Smartphone-Kamera verglichen und bewertet. Als letzter Teil der vorliegenden Arbeit wird eine App als Prototyp entwickelt, die zeigt, wie eine fertige Anwendung mit den zu implementierenden Algorithmen aussehen könnte.

## 1.3 Struktur der Arbeit

Zunächst werden ein paar allgemeine Grundlagen betrachtet, die für das Verständnis dieses Themas unmittelbar von Bedeutung sind. Bevor es dann um die Entwicklung der einzelnen Teile dieser Arbeit geht, wird ein kleiner Blick auf bereits existierende Lösungen zu diesem Thema geworfen. Ein besonderes Augenmerk liegt dabei auf den mobilen Anwendungen, da diese einerseits in direkter Konkurrenz zu der hier zu entwickelnden App stehen, andererseits aber auch eine Quelle an Inspiration bieten können. Gewappnet

mit diesen Eindrücken werden die Anforderungen für diese App aufgestellt. Aus diesen Anforderungen wird dann das Konzept entwickelt, welches für die zu entwickelnden Methoden und Implementierungen genutzt wird. Bevor es dann um das Herzstück dieser Arbeit, der Entwicklung der Algorithmen, geht, werden noch die dafür wichtigen Grundlagen besprochen. Anschließend folgt die Implementierung einer mobilen Anwendung in Android. Zum Schluss folgt eine Zusammenfassung und kritische Betrachtung der Arbeit. Auch wird ein Ausblick auf Ansätze zur Weiterentwicklung gegeben.

# 2 Grundlagen

Zunächst einmal sind an dieser Stelle einige physikalische und technische Grundlagen zu betrachten. Als Erstes wird auf die Grundlagen zur App-Programmierung eingegangen, bevor es um die Bildverarbeitung und abschließend um die Flugbahn von Bällen geht.

## 2.1 Grundlagen zur Programmierung in Android

Android ist eins der zwei gängigen Betriebssystemen für Smartphones. In Deutschland hat Android zur Zeit einen stabilen Marktanteil von 70 bis 80 Prozent, wobei vor allem Geräte im unteren und mittleren Preissegment verkauft werden[1]. Für Android können mit geringem Aufwand selber Anwendungen geschrieben werden. Alles, was dazu benötigt wird, ist ein Android-Smartphone (inklusive Datenkabel) und ein Rechner mit einem der gängigen Betriebssysteme. Zudem wird eine Entwicklungsumgebung benötigt, in der Android-Apps programmiert werden können. Dazu bietet sich vor allem die eigens zu diesem Zweck von Google bereitgestellte und freie Umgebung Android-Studio an[2].

Bei der Programmierung in Android ist auf die Android-Version zu achten, auf der die App laufen soll. Bis zum Entstehungszeitpunkt dieser Arbeit gibt es bereits Android-Version 10. Die entwickelten Apps sind aufwärts- aber nicht abwärtskompatibel. Das bedeutet, dass eine Anwendung, die für Android 8 programmiert ist, auch auf der Betriebsversion Android 10, nicht aber auf Android 6 läuft.[1, S. 10]

Für das Programmieren von Android-Anwendungen werden sogenannte Activities genutzt. Eine Activity besteht aus einem Layout-File, welches das Design der Activity festlegt und einem Java-File (wahlweise auch Kotlin), welches das Verhalten der Activity steuert. Beim Layout-File handelt es sich um eine XML-Datei, die in Android-Studio

[1] Quelle: https://de.statista.com/statistik/daten/studie/184332/umfrage/marktanteil-der-mobilen-betriebssysteme-in-deutschland-seit-2009/ [Letzter Zugriff: 22.02.2021]

[2] Quelle: https://developer.android.com/studio [Letzter Zugriff: 22.02.2021]

geschrieben, oder über einen grafischen Editor bearbeitet werden kann. In ihr werden Informationen über die grafische Darstellung dieser Activity, etwa die Lage und Farbe von Textfeldern und Button, gespeichert. [1, S. 12]

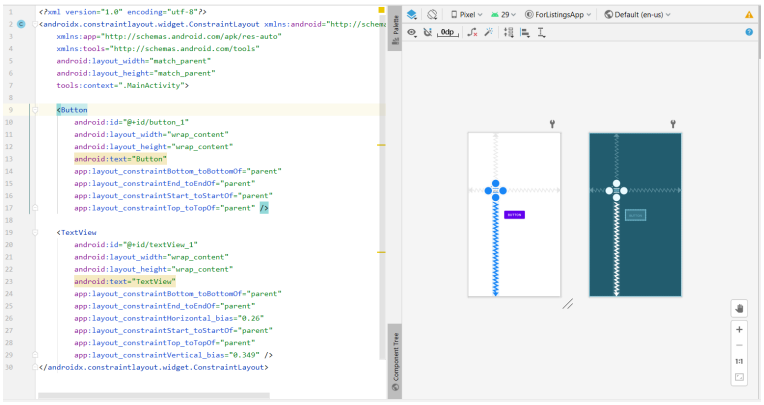


Abbildung 2.1: Bearbeiten der XML-Datei

Bei der Java-Datei handelt es sich um den Quelltext, in dem festgelegt ist, was beim Klicken eines Button oder beim Aufrufen der Activity passiert. Damit das System damit interagieren kann, ist die Programmierung in einer Klasse zu realisieren. Diese erbt für die Interaktion[3] notwendige Methoden von der Superklasse „AppCompatActivity“.

```
1  public class MainActivity extends AppCompatActivity{
2        ...
3  }
```

Listing 2.1: Main Activity

Die Besonderheit der Programmierung mobiler Anwendungen in Android ist es, dass sich diese stark nach dem Lebenszyklus einer App richtet. Dieser Lebenszyklus, der in Abbildung 2.3 dargestellt ist, richtet sich nach den Stadien, in der sich eine App [4] beim Benutzen befinden kann. Bei Eintritt in ein Stadium wird die entsprechende Methode aufgerufen. Diese kann im Java-File programmiert werden. Wird die App beispielsweise aufgerufen, so wird zunächst die „onCreate-Methode“ aufgerufen. Wenn dann der Nutzer beispielsweise zu einer anderen App wechselt, dann wird die „onStop-Methode“ aufgerufen. [1, S. 167]

[3] Etwa Methoden für den Lebenszyklus der App.
[4] Genauer gesagt die entsprechende Activity, aus denen die App besteht.

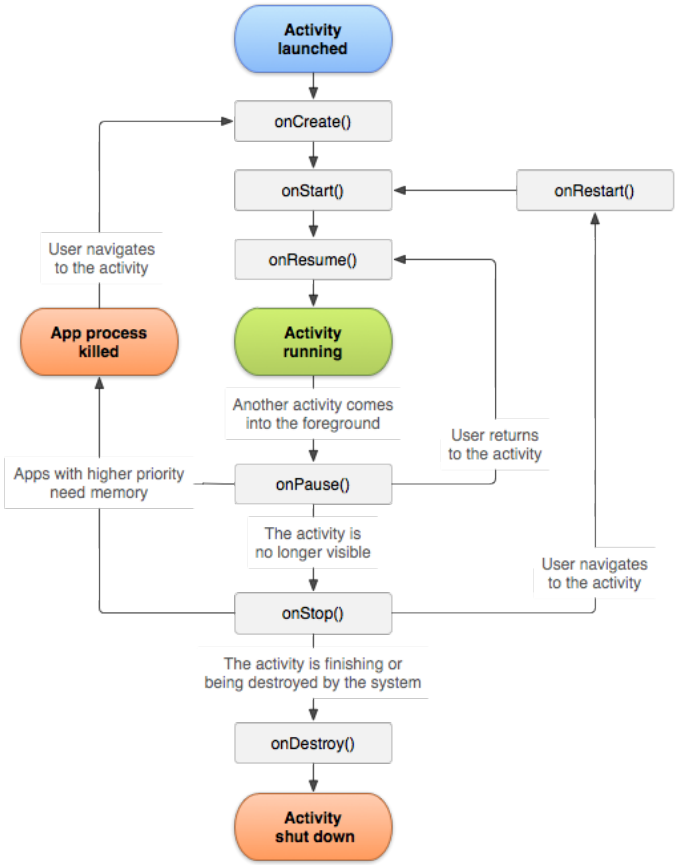


Abbildung 2.2: Lebenszyklus einer App[5]

Durch das Erben von der Superklasse sind alle Methoden bereits vorhanden und müssen nicht selber implementiert werden. Soll eine dieser Methoden verändert und durch eigenen Quelltext ergänzt werden, so muss diese überschrieben werden. Dies wird mit der Zeile „@Override“ angekündigt, wie es beispielhaft im Quelltext-Auszug 2.2 zu sehen ist.

```
1  @Override
2  protected void onDestroy() {
3      super.onDestroy();
4      ...
5  }
```

Listing 2.2: Methoden überschreiben

[5]Quelle: https://developer.android.com/guide/components/activities/activity-lifecycle [Letzter Zugriff: 11.01.2021]

Das Implementieren der „onCreate-Methode“ ist notwendig, da in dieser das XML-File geladen wird. Im Anschluss daran können dann Button initialisiert, Daten geladen oder weitere Initialisierungen vorgenommen werden.

```
1  public class MainActivity extends AppCompatActivity {
2
3      TextView tv;
4      Button btn;
5
6      @Override
7      protected void onCreate(Bundle savedInstanceState) {
8          super.onCreate(savedInstanceState);
9          setContentView(R.layout.activity_main);
10
11         tv = findViewById( R.id.textView_1 );
12         btn = findViewById( R.id.button_1 );
13
14         btn.setOnClickListener(new View.OnClickListener() {
15             @Override
16             public void onClick(View view) {
17                 tv.setText( "Hello World" );
18             }
19         } );
20
21     }
22 }
```

Listing 2.3: onCreate-Methode

Das Beispiel in Quelltext-Auszug 2.3 zeigt die „onCreate-Methode“ einer App, die nach dem Klicken auf einen Button den Text „Hello World“ in ein Textfeld ausgibt. Mit der Funktion „setContentView(R.layout.ACTIVITYNAME)“ wird die Verbindung zwischen XML und Java hergestellt. Mit der „findViewById( R.id.ID )-Funktion“ können von den jeweiligen Komponenten Instanzen in Java erzeugt werden. Zuletzt ist dem Button noch ein „View.OnClickListener()“ hinzuzufügen. Dieser wird aktiviert, sobald ein Klick auf den Button stattfindet, sodass der darin befindliche Quelltext ausgeführt wird.

Bei den Lebenszyklus-Methoden und der „onClick-Methode“ beim Button handelt es sich um sogenannte Callbacks. Das sind Methoden aus Superklassen oder Interfaces, die überschrieben werden können und vom System aufgerufen werden. Beim Button beispielsweise wird dem System der „View.OnClickListener()“ übergeben, sodass dieses die darin überschriebene „onClick-Methode“ aufrufen kann.

Beim Design des Layouts einer App werden zwei verschiedene Ausrichtungen des Smartphones unterschieden. Befindet sich das Smartphone hochkant in der Hand des Nutzers, als würde dieser eine Portrait-Aufnahme einer anderen Person machen, so ist vom Portrait-Mode die Rede. Bei einer auf der Seite liegenden Haltung, so als würde der Nutzer eine Landschaft einfangen wollen, spricht man vom Landscape-Mode.

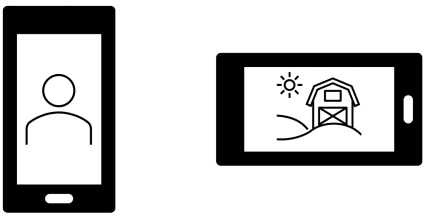

Abbildung 2.3: Smartphone Ausrichtungen

### 2.1.1 Kamerainterfaces

Ein wichtiger Bestandteil dieser Arbeit stellt der Umgang mit der, im Smartphone befindlichen Kamera dar. Es gibt vier Möglichkeiten, wie auf die Bilddaten der Kamera zugegriffen werden kann. Diese werden an dieser Stelle vorgestellt und verglichen. Die Möglichkeiten, die in Android Studio integriert sind, sind drei verschiedene Anwendungsprogrammierschnittstellen (API). Diese Schnittstellen werden Camera, Camera2 und CameraX genannt. Allerdings gilt die Camera API als veraltet. Aus diesem Grund wird sie von vornherein für die weitere Betrachtung ausgeschlossen[6]. Eine weitere Möglichkeit auf die Kameradaten zuzugreifen bietet ein OpenCv-Kamerainterface für Android.

Zunächst wird auf die Camera2 API eingegangen. Um diese verwenden zu können, muss eine relativ aufwendige Struktur im Quelltext geschaffen werden. Zunächst einmal muss ein „TextureView“ in das Layout eingefügt werden, in dem später die Preview angezeigt werden kann. Zudem wird eine Reihe verschiedener Klassen und Interfaces benötigt, die über das System kommunizieren können. Dies ist zum Beispiel dann nötig, wenn nach der Initialisierung das System erst die Bereitschaft zurückmelden muss. Dies wird über Callbacks realisiert. So muss beispielsweise nach der Initialisierung des „TextureViews“ abgewartet werden, bis dieser für weitere Bearbeitungen verfügbar ist. Dazu wird diesem

[6] Quelle: https://developer.android.com/guide/topics/media/camera [Letzter Zugriff: 11.01.2021]

ein „SurfaceTextureListener“ übergeben. Ist die Initialisierung abgeschlossen, so wird in diesem Listener die Methode „onSurfaceTextureAvailable“ aufgerufen, in der dann die weiteren Schritte stattfinden können. Das Camera2-Interface bietet die Möglichkeit eine rein im Hintergrund stattfindende Kamera-Vorschau zu implementieren. Auch können so Videos aufgenommen und abgespeichert werden. Was hingegen nicht möglich ist, ist ein Zugriff auf die geladenen und in der Vorschau gezeigten Bilddaten. Die Abbildung 2.4 gibt den Ablauf der Initialisierung des APIs wieder. Grün dargestellt sind dabei die Schritte, die vom Programmierer im Quelltext vorgenommen werden müssen, blau die, die vom System im Hintergrund ausgeführt werden[7].

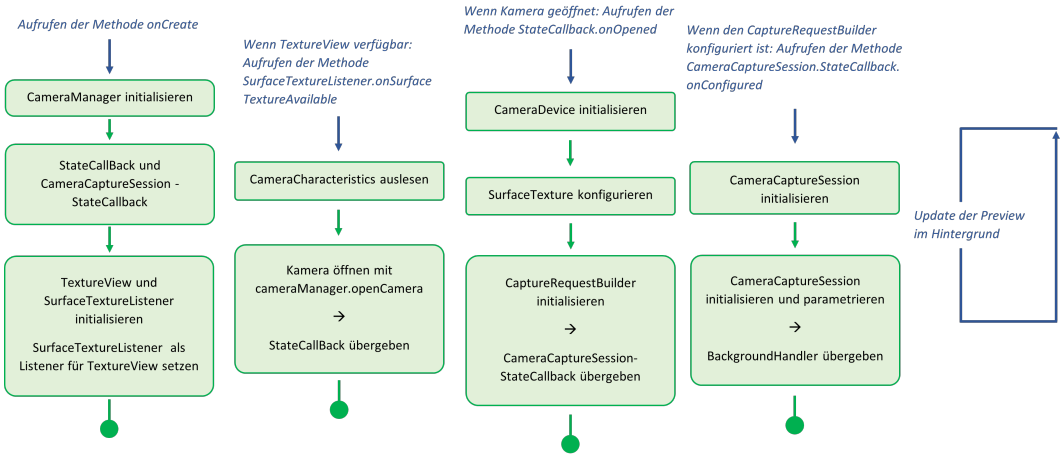


Abbildung 2.4: Initialisierung Camera2 API

Ein Interface, das einen Zugriff bietet, ist die OpenCV-Camera für Android. Um dieses nutzen zu können, müssen zusätzliche Bibliotheken in Android Studio hinzugefügt werden. Diese stehen auf der offiziellen Internetseite von OpenCV zur Verfügung[8] und müssen in Android Studio als neues Modul hinzugefügt werden. Für dieses Kamerainterface muss die MainActivity das Interface „CameraBridgeViewBase.CvCameraViewListener2“ implementieren. Zusätzlich muss für das, in der XML-Datei einzufügende, „JavaCameraView“ eine Instanz der MainActivity erzeugt werden. Sobald neue Bilddaten zur Verfügung stehen, wird fortlaufend die Methode „onCameraFrame“ aufgerufen, der die Bilddaten übergeben werden. Innerhalb dieser Funktion können die Bilddaten bearbeitet und letztendlich zurückgegeben werden, sodass diese über die „JavaCameraView“ auf dem Display

[7] Quelle: https://developer.android.com/training/camerax/vendor-extensions [Letzter Zugriff: 22.02.2021]
[8] https://opencv.org/releases/

ausgegeben werden. Der Quelltext-Auszug 2.4 zeigt diese Funktion.

```
1  @Override
2  public Mat onCameraFrame(CameraBridgeViewBase.CvCameraViewFrame inputFrame) {
3        Mat frame = inputFrame.rgba();
4        ...
5        return frame;
6  }
```

Listing 2.4: onCameraFrame

Dieses Interface bietet die Möglichkeit direkt auf die Bilddaten zuzugreifen und diese zu bearbeiten. Es ist allerdings zu bemerken, dass jegliches Bearbeiten das Aufrufen des nächsten Bildes verzögert und somit direkte Auswirkungen auf die Bildrate hat.

Als dritte Möglichkeit bietet sich die CameraX API an. Diese ist laut Android Developers[9] speziell für die Echtzeitverarbeitung von Bilddaten geschaffen worden. Als eigens von Android entwickelte API sind zum Nutzen von CameraX keine weiteren Bibliotheken einzubinden. Im Kern basiert CameraX auf der Camera2 API, ist für den Nutzer aber wesentlich einfacher zu implementieren und bietet die Möglichkeit auf die aktuell geladenen Bilder zuzugreifen. Es ist eine Instanz eines „ProcessCameraProvider“ zu erzeugen. Dieser Instanz wird über die „addListener“ Methode ein Runable übergeben, in welchen im Hintergrund die weiteren Schritte ausgeführt werden. In den weiteren Schritten werden verschiedene Klassen initialisiert, die das Verhalten der Kameraimplementierung definieren. Dazu gehört die Preview, die Wahl der Kamera oder optional die Analyse der Bilder. Diese Klassen werden dann, wie im Quelltext-Auszug 2.5 zu sehen, zu einem sogenannten Lebenszyklus der Kamera zusammengefasst[10]. Die folgenden Quelltext-Auszüge 2.5 und 2.6 der CameraX API sind in Kotlin geschrieben.

```
1  val cameraProviderFuture = ProcessCameraProvider.getInstance(this)
2
3  cameraProviderFuture.addListener(Runnable {
4        val cameraProvider: ProcessCameraProvider = cameraProviderFuture.get()
5
6        val preview = Preview.Builder().build().also {
7              it.setSurfaceProvider(viewFinder.createSurfaceProvider())
8           }
9
10       val imageAnalyzer = ImageAnalysis.Builder().build().also {
```

[9] Quelle: https://developer.android.com/training/camerax [Letzter Zugriff: 22.02.2021]
[10] Quelle: https://codelabs.developers.google.com/codelabs/camerax-getting-started[Letzter Zugriff: 22.02.2021]

```
11                  it.setAnalyzer(cameraExecutor, MyAnalyzer() )
12          }
13
14      val cameraSelector = CameraSelector.DEFAULT_BACK_CAMERA
15
16          try {
17                  cameraProvider.unbindAll()
18                  cameraProvider.bindToLifecycle(
19                      this, cameraSelector, preview, imageCapture, imageAnalyzer)
20
21           } catch(exc: Exception) { ... }
22
23  }, ContextCompat.getMainExecutor(this))
```

Listing 2.5: Initialisierung der CameraX API

Die Analyse der Bilder erfolgt durch eine Klasse „ImageAnalysis.Analyzer“. In dieser ist die Methode „analyze“ zu überschreiben, die für jedes Bild aufgerufen wird und der die Bilddaten übergeben werden, wie der Auszug 2.6 zeigt.

```
1   private class LuminosityAnalyzer() : ImageAnalysis.Analyzer {
2
3       ...
4
5           override fun analyze(image: ImageProxy) {
6
7             val buffer = image.planes[0].buffer
8             val data = buffer.toByteArray()
9             val pixels = data.map { it.toInt() and 0xFF }
10            val luma = pixels.average()
11
12            ...
13
14            image.close()
15          }
16
17  }
```

Listing 2.6: Analyseklasse der CameraX API

### 2.1.2 Einbinden nativer Bibliotheken

Über Android-Studio lassen sich in C++ geschriebene Funktionen in die App einbinden. Dazu ist in Android-Studio beim Erstellen des Projektes der „Native C++“ Typ auszuwählen, wie in Abbildung 2.5 aus Android-Studio zu sehen ist.

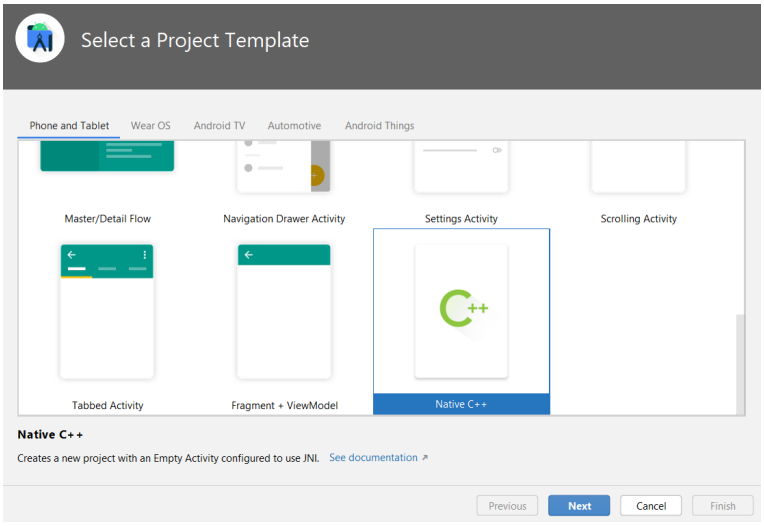


Abbildung 2.5: Projektauswahl in Android-Studio

Es wird von Android-Studio eine Ordnerstruktur angelegt, in der die Bibliotheken gespeichert werden (siehe Abbildung 2.6 links). Im Quelltext müssen die verwendeten Bibliotheken dann in einem statischen Block angekündigt werden, wie in Abbildung 2.6 rechts zu sehen ist.

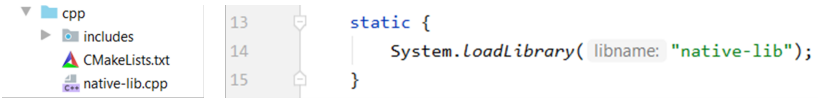


Abbildung 2.6: Einbinden einer nativen Bibliothek

Zusätzlich müssen die Funktionen, die verwendet werden als native Funktionen angekündigt werden. Dies geschieht am Ende der Klasse in der diese verwendet werden, wie in Abbildung 2.7 zu sehen ist.

```
110     public native String stringFromJNI();
111     public native int addNumbers(int num1, int num2);
112     public native int subNumbers(int num1, int num2);
113     public native double difNumbers(int num1, int num2);
114     public native double mulNumbers(int num1, int num2);
```

Abbildung 2.7: Ankündigung nativer Funktionen

Abbildung 2.8 zeigt die Implementierung einer dieser Funktionen in der Bibliothek. Bei der Rückgabe von Parametern kann es dazu kommen, dass diese umgewandelt werden müssen, da diese in Java und C++ anders referenziert werden. Wie außerdem zu sehen ist, können zusätzliche Bibliotheken für die Nutzung in der C++-Bibliothek eingebunden werden. Auch die OpenCV-Bibliothek kann so verwendet werden, um Bilddaten zu verarbeiten, die den nativen Funktionen übergeben werden.

```
1  #include <jni.h>
2  #include <string>
3
4  extern "C" JNIEXPORT jstring JNICALL
5  Java_com_example_bachelor13_MainActivity_stringFromJNI( JNIEnv* env, jobject MainActivity ) {
6      std::string hello = "Hello from C++";
7      return env->NewStringUTF(hello.c_str());
8  }
```

Abbildung 2.8: Native Funktionen

## 2.2 Bildverarbeitung

In der Bildverarbeitung geht es um die Bearbeitung, Verarbeitung und Auswertung von Bilddaten. Als Bilddaten wird dabei die Computer-interne Repräsentation des digitalen Bildes verstanden, welche aus den Farbwerten der einzelnen Pixel besteht. Diese werden in Form eines dreidimensionalen Vektors der Größe $X \cdot Y \cdot 3$ gespeichert. $X$ und $Y$ stehen dabei für die Anzahl der Pixel in horizontaler ($X$) und vertikaler ($Y$) Richtung. Jedes Pixel besteht aus 3 Werten, die die Farben Rot, Grün und Blau darstellen. Diese können Werte zwischen 0 und 255 annehmen, wobei 0 für die minimale und 255 für die maximale Intensität steht.

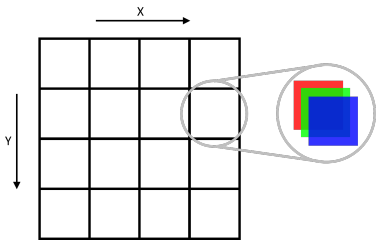


Abbildung 2.9: Aufbau von Bilddaten

Die linke obere Ecke der Bilddaten wird mit den Koordinaten $(x, y) = (0, 0)$ referenziert; für die weiteren Ecken gilt: $(X, 0)$ für die Ecke rechts oben, $(0, Y)$ für die Ecke links unten und $(X, Y)$ für die Ecke rechts unten. Dabei ist festzuhalten, dass höhere Werte für $y$, Koordinaten weiter unten im Bild referenzieren und die Werte für $x$ nach rechts hin ansteigend sind. [3, S. 10,11]

Für die Aufnahme von Videos gibt es momentan drei gängige Formate: HD (1280x720), Full HD (1920x1080) und 4K UHD (3840x2160). Dabei gilt Full HD, auch als 1080p bezeichnet, als Standard.

### 2.2.1 OpenCV für Python

OpenCV ist eine Open Source Bibliothek, die Funktionen und Klassen zur Bildverarbeitung bietet. Die Entwicklung wurde im Jahr 2000 von Intel gestartet[11]. Mittlerweile ist OpenCV die gängige Wahl bei der Entwicklung von Bilderverarbeitungsanwendungen. OpenCV steht für mehrere Programmiersprachen zur Verfügung, unter anderem auch C++ und Python und ist auch in der Android-Programmierung verfügbar. Diese Bibliothek verfügt über eine Reihe von Funktionen, die für die gewünschte Implementierung benutzt werden könnten. [2]

### 2.2.2 Differenzbilder

Unter anderem bietet OpenCV die Möglichkeit ein Differenzbild zu berechnen. Bei Differenzbildern wird aus zwei Bildern derselben Dimension ein neues Bild erzeugt, wobei jedem Pixel der Absolutwert der Differenz der Pixelwerte der zwei ursprünglichen Bilder

[11] Quelle: https://de.wikipedia.org/wiki/OpenCV [Letzter Zugriff: 23.02.2021]

zugewiesen wird. Daraus lassen sich unter anderem Bewegungen erkennen, die zwischen diesen beiden Bildern stattgefunden haben. Der Aufruf dieser Funktion in Python sieht wie in Auszug 2.7 zu sehen aus:

```
1  difference = cv2.absdiff( frame1 , frame2 )
```

Listing 2.7: Berechnung des Differenzbildes

Bewegt sich ein Objekt, etwa ein Ball, und tritt diese auf den Bildern „frame1“ und „frame2“ an unterschiedlichen Stellen auf, so ist es im Differenzbild zwei mal zu sehen. Dies zeigt Abbildung 2.10 am Beispiel eines Tennisballs.

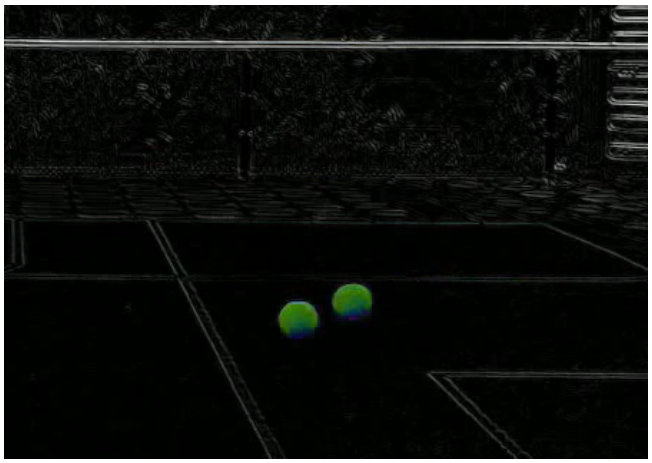

Abbildung 2.10: Doppelerscheinung eines Tennisballs

Die Differenzbilder können auch für die Analyse von Videos verwendet werden. Dabei wird jeweils zwischen zwei aufeinanderfolgenden Bildern das Differenzbild berechnet. Der Ablauf der Verarbeitung läuft dabei grundsätzlich ab, wie im Quelltext-Auszug 2.8 sehen:

```
1  lastFrame = None
2
3  while (videoLoad.isOpened()):
4        ret, frame = videoLoad.read()
5
6        if lastFrame is None:
7              lastFrame = frame
8              continue
```

```
9
10          difference = cv2.absdiff( frame , lastFrame )
11
12          # weitere Verarbeitungen
13
14          lastFrame = frame
```

Listing 2.8: Differenzbilder in der Videoverarbeitung

Zunächst wird das aktuelle Bild geladen (Zeile 4), um aus diesem und dem vorherigen Bild das Differenzbild zu berechnen (Zeile 10). Mit diesem können weitere Verarbeitungen stattfinden. Zum Ende jedes Schleifendurchganges wird das aktuelle Bild für den nächsten Durchlauf in die Variable „lastFrame“gespeichert (Zeile 14). Im ersten Schleifendurchlauf, in dem es noch kein vorheriges Bild gibt, wird die Berechnung übersprungen (Zeilen 6 bis 8).

## 2.3 Flugbahn eines Balles

Ein Fußball, der beim Profisport vom Elfmeterpunkt geschossen wird, erreicht Geschwindigkeiten bis zu 120 km/h[12]. Das entspricht etwa 33.3 Meter pro Sekunde.

Die Flugbahn von Bällen kann unter Vernachlässigung des Luftwiederstandes durch eine Parabel beschrieben werden. Die dafür benötigte Funktion wird als Wurfparabel bezeichnet[13].

$$y(x) = \tan(\alpha) \cdot x + \frac{g}{2 \cdot v_0^2 \cdot \cos^2(\alpha)} \tag{2.1}$$

Dabei ist $y$ die Flughöhe in Abhängigkeit der Flugweite $x$. Die Variable $v_0$ stellt die Anfangsgeschwindigkeit des Balles dar und $g$ die Erdbeschleunigung von $9,81 \frac{m}{s^2}$. Dabei ist jedoch der Luftwiderstand, der mit zunehmender Geschwindigkeit des Balles größer

[12] Quelle: https://www.simplyscience.ch/teens-liesnach-archiv/articles/wie-schnell-fliegt-ein-fussball-im-durchschnitt-ins-tor.html [Letzter Zugriff: 22.02.2021]

[13] Quelle: https://www.weltderphysik.de/thema/hinter-den-dingen/flug-ohne-luftwiderstand/ [Letzter Zugriff: 22.02.2021]

wird, nicht mit berechnet worden. Fließt dieser mit ein, so ist von einer ballistischen Kurve die Rede[14]. Diese ist stark von der Wurfparabel abweichend[15].

$$y(x) = x\left(\tan(\alpha) + \frac{m \cdot g}{\beta \cdot v_0 \cdot \cos(\alpha)}\right) + \frac{m^2 \cdot g}{\beta^2} \cdot ln\left(1 - \frac{\beta}{m} \cdot \frac{x}{v_0 \cdot \cos(\alpha)}\right) \tag{2.2}$$

Dabei steht $\beta$, nach dem Gesetz von Stokes, für das Produkt aus $6\pi$, dem Radius des Balles und der „dynamischen Viskosität des Fluids, in dem sich das Partikel befindet“[16]. Letzteres bezieht sich auf die Luft, durch die sich der Ball bewegt und beträgt bei $20°C$ Umgebungstemperatur etwa $17,2\mu Pa \cdot s$[17]. Weitere wichtige Variablen sind die Ballmasse $m$, die Anfangsgeschwindigkeit $v_0$, die Erdbeschleunigung $g$ und der Abwurfwinkel $\alpha$. Letzterer entspricht dem Schusswinkel, der in Kapitel 5.1.1 besprochen wird.

[14] Quelle: https://www.wissen.de/lexikon/ballistische-kurve [Letzter Zugriff: 22.02.2021]
[15] Quelle: https://www.weltderphysik.de/thema/hinter-den-dingen/rolle-des-luftwiderstands/ [Letzter Zugriff: 22.02.2021]
[16] Quelle: https://de.wikipedia.org/wiki/Gesetz_von_Stokes [Letzter Zugriff: 22.02.2021]
[17] Quelle: https://www.flottweg.com/de/wiki/trenntechnik/dynamische-viskositaet/ [Letzter Zugriff: 22.02.2021]

# 3 Stand der Technik

Da es zur Bewertung der Funktionsfähigkeit wichtig ist, vielversprechende existierende Lösungen zu kennen, werden diese an dieser Stelle betrachtet. Als Erstes wird dabei auf eine hardwaretechnische Lösung eingegangen. Da das Ziel der vorliegenden Arbeit jedoch eine rein softwaretechnische Implementierung ist, liegt der Fokus in diesem Kapitel auch auf Lösungen in diesem Bereich.

## 3.1 Radar-Messgerät

Das an dieser Stelle betrachtete Radar-Messgerät ist, wie bereits erwähnt, die Inspiration für die vorliegende Bachelorarbeit. Es handelt sich um das „SmartPro“ von „Net Playz“[1]. Dieses Gerät funktioniert mittels Radarmessung. Der Spieler muss den Ball in Richtung des Gerätes schießen, wie in der Abbildung 3.1 aus der Benutzeranleitung zu entnehmen ist.

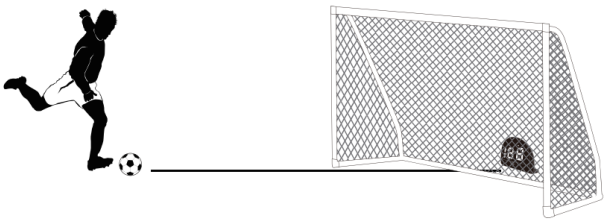

Abbildung 3.1: Auszug aus der Benutzeranleitung[2]

[1] SmartPro Personal Sports Radar (Modelbezeichnung: ODIS-06-R1) von Netz Plays. Siehe: http://www.netplayzsport.com/product.php?id=35 [Letzter Zugriff: 18.02.2021]

[2] Quelle: User Manual Net Palyz für das Modell ODIS-06-R1. Die Benutzeranleitung ist zum Download verfügbar unter: https://www.trigreatusa.com/product_page?id=10&lang=en [Letzter Zugriff: 06.03.2021].

Die optimale Entfernung beim Messen von Fußbällen und Tennisbällen liegt laut Benutzeranleitung zwischen 6 und 15 Metern, wobei je nach Einstellung und Sportart Geschwindigkeiten zwischen 5 und 199 $km/h$ gemessen werden können. Über die Messweise wird in der Anleitung die Angabe gemacht, dass die relative Geschwindigkeit eines Objektes in Richtung des Messgerätes gemessen wird. Ferner wird erwähnt, dass die Geschwindigkeit nur bei direkter Bewegung auf das Messgerät zu korrekt gemessen wird. Dies ist der Fall, da bei größer werdendem Winkel auch die Abweichungen in der Messung zunehmen. Es wird also lediglich der Teil der Bewegung gemessen, der direkt auf das Messgerät gerichtet ist. Es stellen sich also Abweichungen, wie in Kapitel 5.2 beschrieben, ein. Bezüglich des Radar-Messgerätes gelten diese Abweichungen nicht nur für den vertikalen, sondern auch für den horizontalen Schusswinkel. Die dadurch zu erwartenden Abweichungen sind durch den Umstand, dass im Gegensatz zur Aufzeichnung durch eine Kamera, in einer Dimension weniger gemessen werden kann, größer als bei der Kamera.


Abbildung 3.2: Radar-Messgerät auf einem Kamerastativ

Das Radar-Messgerät ist einfach zu bedienen und bietet relativ genaue Messergebnisse. Es verfügt alles in einem über eine gute User-Experience. Auf der anderen Seite ist es eher kostenintensiv und stellt mit derzeit 60 bis 190 Euro (für dieses Modell, online bestellbar) die teuerste hier aufgezeigte Lösung dar.

## 3.2 Mobile Anwendungen

Sowohl für iOS, als auch für Android gibt es bereits verschiedene Anwendungen, die entweder genau für den Zweck der Messung von Ballgeschwindigkeiten entwickelt worden sind, oder zu mindestens auch dafür eingesetzt werden können. Im Folgenden wird auf diese eingegangen und es werden Vor- bzw. Nachteile dieser Apps herausgestellt.

### Radar Gun

Den Anfang macht eine Reihe von Apps mit Namen wie „Speed Gun“, „Radar-Gun[3]“, oder „Geschwindigkeitsmesser“, die sich in ihrer Funktionsweise ähnlich sind und deswegen hier zusammengefasst werden können. Diese Apps verfügen über eine Kamera-Vorschau und einen Curser, der mit dem Finger bewegt werden kann. Eingegeben werden muss die mittlere Entfernung zu dem zu messenden Objekt. Zieht man den Finger über den Bildschirm, so wird daraus die Geschwindigkeit berechnet. Zu diesem Zweck müsste auch die Brennweite bzw. der Bildwinkel der Kamera bekannt sein, was keine Schwierigkeit darstellt, da diese, wie bereits erwähnt, für die Hauptkameras der Smartphones ähnlich sind.
Der Vorteil dieser Apps liegt klar in der Einfachheit. Sie können leicht bedient werden, und bieten zu mindestens das Potential die Geschwindigkeit eines Balles zu messen. Nach genauerem Testen hat sich allerdings gezeigt, dass es sehr umständlich sein kann, ein kleines sehr schnelles Objekt wie einen Ball nachzuverfolgen, gerade weil die Stelle, an der sich der Ball auf dem Bildschirm befindet durch den Finger verdeckt wird. Außerdem stellt das Bewegen des Fingers über den Bildschirm nach einiger Zeit eine unkomfortable Bewegung dar.

[3] Etwa: https://play.google.com/store/apps/details?id=kr.sira.speed[Letzter Zugriff: 18.02.2021]

### Speed Clock

Diese App ist nur für iOS verfügbar und ist dort nur käuflich zu erwerben. Aus diesem Grund konnte die App nicht ausführlich getestet werden. Es standen dennoch einige Promotion-Videos der Vertreibenden zur Verfügung, an denen der Nutzen dieser App abgeschätzt werden kann. Diese App bietet zwei mögliche Funktionsweisen, die für die Messung von Ballgeschwindigkeiten genutzt werden können.
Die erste Möglichkeit besteht darin, dass über Bilderkennung, Eintritt und Austritt eines Objektes in den Bildausschnitt erkannt werden. Aus der Zeitdifferenz und der einzugebenden mittleren Entfernung wird die Geschwindigkeit berechnet, die das Objekt gehabt haben muss. An dieser Stelle wären genauere Tests mit der App sehr hilfreich gewesen, um herauszufinden, wie die Detektion erfolgt, und wie robust diese ist. Die vorliegenden Videos erwecken den Anschein, dass die Detektion über eine Bewegungserkennung funktioniert und, dass das Smartphone zu diesem Zweck fest installiert sein muss. Die Hersteller dieser App empfehlen diese Methode jedoch nicht explizit zur Messung von Ballgeschwindigkeiten[4].
Zu diesem Zweck wird eine andere Funktion empfohlen, bei der für die Messung vom Nutzer ein kurzes Video aufgezeichnet werden muss. Dieses Video ist dann abzuspielen und der Benutzer muss Start- und Endposition des Balles markieren. Aus diesen Positionen zusammen mit der mittleren Entfernung der Kamera zum Ball, kann die Geschwindigkeit des Balles berechnet werden.
Über die Vor- und Nachteile kann an dieser Stelle nur gemutmaßt werden, da diese auch von der Performance dieser App abhängen. Generell lässt sich aber sagen, dass das Markieren der Ballposition mit den Fingern keine Alternative Möglichkeit für diese Arbeit darstellt, da hier eine automatische Detektion der Flugbahn das Ziel ist.

### Velo

Velo[5] ist eine speziell für die Messung von Baseballwürfen entwickelte App. Diese ist für beide Plattformen verfügbar, wobei sich beide Apps vom Aussehen deutlich unterscheiden. Zum Zeitpunkt der Bachelorarbeit war Velo im App-Store nicht verfügbar, weswegen nur die Android-Version getestet werden konnte. In dieser App gibt es eine Kameravorschau, mit einem kleinen Ausschnitt, innerhalb dessen sich der Ball während des Fluges

[4] Quelle: https://www.youtube.com/watch?v=kfUAIbqQtbE [Letzter Zugriff: 14.02.2021]
[5] Siehe: https://play.google.com/store/apps/details?id=com.engee.Velo [Letzter Zugriff: 18.02.2021]

befinden muss. Mit der zuvor eingestellten Entfernung zur Wurflinie kann die Geschwindigkeit berechnet werden. Die Erkennung des Balles innerhalb dieses Feldes scheint über eine Bewegungserkennung zu funktionieren. Dies legen Aussagen der Hersteller nahe[6]. Laut Hersteller kann die App zum Messen in der Hand gehalten werden. Doch bei Tests, wie stark das Bild verwackelt werden darf, hat sich ergeben, dass schon bei kleinsten Bewegungen das Ergebnis falsch ist.
Velo besticht mit der Einfachheit, mit der diese App zu nutzen ist. Nachteilig ist, dass diese App sehr ungenau ist, wenn sie nicht fest montiert ist, oder im Hintergrund noch andere Bewegungen stattfinden. Das allgemeine Konzept dieser App ist jedoch sehr ansprechend.

[6]Quelle: https://www.youtube.com/watch?v=sAaj_J-ypi8 [Letzter Zugriff: 14.02.2021]

# 4 Anforderungsanalyse

Für die Entwicklung eines Produktes ist es unerlässlich eine genaue Vorstellung zu haben, wer dieses Produkt benutzen wird, wer bei der Entwicklung beteiligt ist und welchen Anforderungen es genügen soll. Auf diese Fragen wird in diesem Kapitel genauer eingegangen. Zunächst einmal wird betrachtet, welche Personen oder Personengruppen bei der Entwicklung zu beachten sind.

## 4.1 Stakeholder

Drei Personen(gruppen) haben ein besonderes direktes oder indirektes Interesse an dieser App und somit an der Entwicklung, die in dieser Arbeit stattfindet. Das sind der Auftraggeber dieser Arbeit, die potenziellen Nutzer einer fertigen App und letztendlich auch die Entwickler und Weiterentwickler, die an dieser Arbeit oder deren Weiterentwicklungen arbeiten werden. In den folgenden Abschnitten wird auf diese Interessensvertreter und ihre Anforderungen eingegangen sowie beschrieben, wie diese im Laufe des Projektes eingebunden werden.

#### Auftraggeber

Prof. Dr. Marc Hensel fungiert in dieser Arbeit als Auftraggeber und Ansprechpartner und wird im Folgenden als Auftraggeber bezeichnet. In mehreren Meetings und Absprachen wurden Anforderungen und gewünschte Ergebnisse kommuniziert. Ein Ergebnis, das für den Auftraggeber von Bedeutung ist, ist eine funktionierende App, oder zumindest die Entstehung eines implementierbaren Konzeptes. Das bedeutet, dass ein funktionsfähiger Prototype der App zusammen mit einer implementierbaren Bildverarbeitung entsteht, die bei Weiterentwicklung zu einer funktionierenden App zusammengefügt werden kann. Des Weiteren, da im Zuge dieser Arbeit kein fertiges Produkt entsteht, sind vor allem zwei Aspekte von Bedeutung: die Weiterverwendbarkeit des entwickelten Quelltextes und

die Aufbereitung und Weitergabe von technologischen Erkenntnissen. Letzteres umfasst vor allem die Frage, wie der Umgang mit der Kamera in Android und die Bildverarbeitung in Android funktioniert. Im Verlauf der Entwicklung wurden immer wieder zu Fragen der Entwicklung oder bei Unklarheiten Absprachen gehalten.

### Die Nutzer

Eine weitere unbedingt zu bedenkende Personengruppe stellen die potenziellen Nutzer dieser App dar. Bei diesen kommt es zu einer gewissen Varianz bei den Anforderungen, je nachdem in welcher Situation sich der Nutzer befindet, oder zu welchem Zweck die App genutzt wird.

Beispielsweise soll diese App primär zum Schusskrafttraining beim Fußball in Kinder- und Jugendvereinen genutzt werden. Damit steht die Genauigkeit der Ergebnisse im Vordergrund, da die Ergebnisse zum Vergleichen und zum Beobachten des Trainingsfortschrittes möglichst genau sein sollten. Es ist auch davon auszugehen, dass eine übersichtliche und einfach zu bedienende App gewünscht ist. Dies ist vor allem dann der Fall, wenn die App von jungen Spielern selber bedient wird, oder wenn beispielsweise das Schusskrafttraining spontan stattfindet und keine Vorbereitungen getroffen wurden. Letzteres bedeutet auch, dass eine Nutzung der App aus der Hand zu ermöglichen ist. Diese Nutzergruppe wurde hier am Beispiel einer Fußballmannschaft beschrieben, lässt sich aber auch auf andere Sportarten problemlos übertragen. So könnte diese Situation genau so beim Tennis- oder Handballtraining entstehen. Eine weitere Anforderung ist, das Vorhandensein einer Spielerverwaltung, die den Trainingsfortschritt speichern und anzeigen kann.

Eine weitere mögliche Nutzergruppe stellen die Organisatoren von Straßenfesten dar. Neben einem Torwandschießen könnte auch ein Wettschießen nach der Schussgeschwindigkeit auf einem Straßenfest als Attraktion oder Gewinnspiel angeboten werden. Auch hier spielt die Genauigkeit eine wichtige Rolle, da nach den Messungen eventuell Preise vergeben werden. Aber im Vordergrund steht, dass die App unkompliziert zu nutzen ist. Für diesen Anwendungsfall wird es nicht nötig sein die App aus der Hand zu bedienen, da für das Straßenfest an sich bereits Aufbauten vorgenommen werden müssen und dies auch für diesen Stand der Fall sein wird. Die App sollte jedoch robust gegenüber Bewegungen im Hintergrund sein, da bei einem Straßenfest damit zu rechnen ist, dass Menschen an diesem Stand vorbeilaufen. Um am Ende des Tages einen Gewinner bestimmen zu kön-

nen, ist es von Vorteil, wenn die App die Tagesrekorde anzeigen oder gar eine Rangliste aller Ergebnisse eines Tages ausgeben kann.

Als dritte mögliche Nutzergruppe werden hier Kinder betrachtet, die diese App als spielerischen Aspekt nutzen wollen. Hier steht im Vordergrund, dass die App einfach zu nutzen und übersichtlich ist. Eine aufwendige Spielerverwaltung ist nicht notwendig, da für diese Zielgruppe der spielerische Aspekt im Vordergrund steht. Der wichtigste Aspekt in diesem Punkt ist jedoch ein robuster Algorithmus. Es ist zu erwarten, dass diese App in diesem Fall ausschließlich ohne zusätzlichen Messaufbau aus der Hand genutzt wird.

Die hier dargestellten Personengruppen wurden durch rein hypothetische Überlegungen erstellt. Grund dafür ist unter anderem die aktuelle Lage durch die andauernde Covid19-Pandemie, die ein Testen der App mit den genannten Personengruppen schwer bis unmöglich macht. Die Interessen dieser Gruppen werden mit eingeplant, indem im Projekt regelmäßig der Fortschritt zusammen mit neu gewonnenen Erkenntnissen mit den hypothetischen Forderungen abgeglichen wird.

#### Entwickler und Weiterentwickler

Die letzten Stakeholder, auf die hier eingegangen wird, sind der Entwickler und die Weiterentwickler, die an diesem Produkt arbeiten werden. Die Anforderungen des Entwicklers sind zum einen, dass am Ende fertige Bildverarbeitungsalgorithmen sowie ein App-Prototyp entstehen und zum anderen, dass die Entwicklungsarbeiten mit, soweit möglich, bekannten Plattformen stattfinden. Letzteres meint, dass sich für die Entwicklung nicht in eine zu große Menge an neuen Plattformen und Programmiersprachen eingearbeitet werden muss. Dies würde viel wertvolle Zeit kosten und den Entwicklungsprozess unnötig teuer machen, wenn ein Wechsel nicht zwingend notwendig ist. Da die Weiterentwickler zu diesem Zeitpunkt noch nicht bekannt sind, kann dieser Punkt für sie nicht mitbedacht werden. Es kann aber davon ausgegangen werden, dass bei Verwendung der gängigen Plattformen und Programmiersprachen dieser Umstand nicht allzu gravierend ausfallen wird, da es sich bei den Weiterentwicklern um Studenten handeln wird, die im Zuge von Projekten oder Abschlussarbeiten an dieser App arbeiten werden. Die fundamentalen Anforderungen der Weiterentwickler sind die Wartbarkeit des Projektes und die Wiederverwendbarkeit des Quelltextes. So ist es wichtig, dass der Quelltext übersichtlich, gut auskommentiert und modular gestaltet wird, damit sich in diesen schnell hineingearbeitet werden kann. Dies deckt sich mit den Interessen des Entwicklers, da dies auch bei

der Entwicklung eine Rolle spielt, um Änderungen schneller und gezielter vornehmen zu können. Auch ist darauf zu achten, dass spätere Versionswechsel von beispielsweise der SDK oder der Android-Version ohne komplette Neuprogrammierung möglich sind. Ein weiterer Punkt für die Weiterentwickler ist eine saubere und ausführliche Dokumentation, anhand derer sich in das Projekt eingearbeitet werden kann. Wie bei den Nutzern werden die Anforderungen der Weiterentwickler in regelmäßigen Abständen hypothetisch abglichen. In den meisten Fällen jedoch stimmen die Anforderungen mit denen des Entwicklers überein, sodass diese permanent bedacht werden.

## 4.2 Anwendungsfälle

Die möglichen Anwendungsfälle variieren je nach Nutzer und Nutzungskontext, in dem die App verwendet wird. Das folgende Anwendungsfalldiagramm gibt eine Übersicht über die Funktionalität, die die App bieten soll.

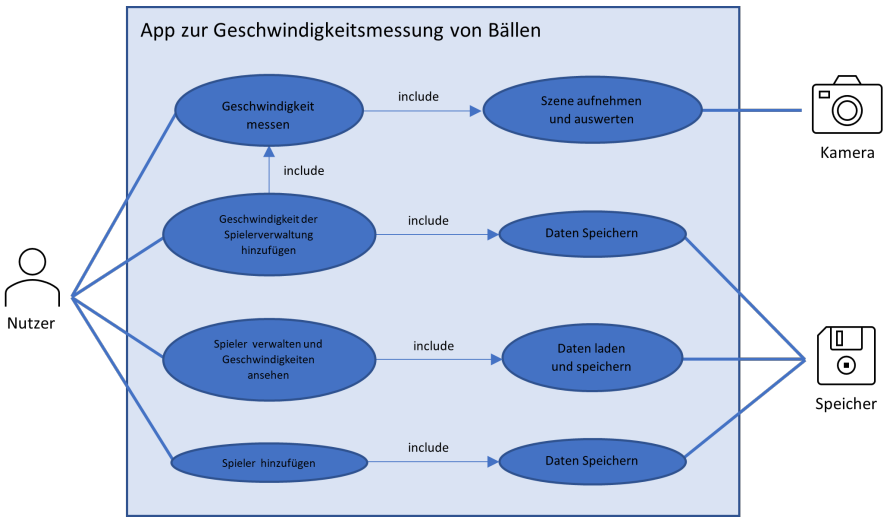


Abbildung 4.1: Anwendungsfalldiagramm

Der Nutzer kann Geschwindigkeiten messen und diese anschließend Spielern zuordnen, sofern eine Spielerverwaltung angelegt worden ist. Für die Spielerverwaltung muss auf den Langzeitspeicher des Smartphones zugegriffen werden, für die Videoaufnahme auf die Kamera.
Generell lassen sich die Anwendungsfälle in zwei Arten einteilen. Zum einen die, bei

denen es sich um ein geplantes Vorhaben handelt und zum anderen die, bei denen es sich eher um eine spontane Nutzung handelt. Bei ersteren können Vorbereitungen getroffen werden. Es kann zum Beispiel eine Halterung für das Smartphone oder eine Abtrennwand zum Abschirmen von Hintergrundbewegungen vorbereitet werden. Es würde also das Smartphone verwacklungsfrei und hauptsächlich von einer Person genutzt werden, die die Schussgeschwindigkeiten anderer misst. Bei der anderen, spontanen Nutzung, würde das Smartphone vor allem aus der Hand verwendet werden.

## 4.3 Anforderungen

Aus den eben besprochenen Aspekten werden funktionale und nicht funktionale Anforderungen erstellt. Die hier aufgelisteten Anforderungen betreffen sowohl die Algorithmen zur Bildverarbeitung, als auch die endgültige App. Zunächst werden die funktionalen Anforderungen betrachtet.

F1 Die App muss für die Messung von Ballgeschwindigkeiten aller gängigen Sportarten verwendbar sein, aber ein besonderes Augenmerk auf den Fußball legen.

F2 Die App muss für Smartphones mit Android 8 verfügbar sein. Diese Anforderung wird vom Auftraggeber gestellt.

F3 Die App muss dem Nutzer die gemessene Geschwindigkeit anzeigen und diesem die Möglichkeit bieten diese der Spielerverwaltung hinzuzufügen.

F4 Die App muss über eine Spieleverwaltung verfügen. Diese Spielerverwaltung muss den Namen der Spieler und die Messergebnisse inklusive Datum enthalten. Zudem muss die Spielerverwaltung die Möglichkeit bieten Spieler hinzuzufügen und zu löschen. Es geht dabei nur um einzelne Spieler, nicht um Teams.

F5 Die App muss aus freier Hand verwendbar sein.

F6 Die App muss den Nutzer über fehlerhafte Messungen informieren und diese verwerfen.

F7 Die App muss über eine Anzeigemöglichkeit für Tagesrekord-Werte verfügen.

Die nicht funktionalen Anforderungen wurden in Anbetracht der Grundlagen und einer hypothetischen Betrachtung des Nutzers entwickelt. Für letzteres wurde die Nutzung der App aus Sicht des Nutzers betrachtet, um so Ansprüche und Forderungen des Nutzers zu formulieren. Die nicht funktionalen Anforderungen sind:

N1 Die App muss intuitiv und einfach zu benutzen sein, damit sich neue Nutzer schnell in die App einfinden und eine möglichst gute User Experience entsteht.

N2 Die Bildverarbeitung muss möglichst schnell funktionieren. Optimalerweise in Echtzeit, oder nach Aufnahme der Szene ohne lange Wartezeiten für den Nutzer. Die dafür maximale Wartezeit für den Nutzer soll 25 Sekunden betragen.

N3 Wenn die App bei der Benutzung abstürzt oder vom System beendet wird, dürfen bis dahin ausgewertete Daten nicht verloren gehen.

N4 Die App muss ein optisch ansprechendes Design haben.

N5 Die App muss Geschwindigkeiten zwischen 10 und 120 km/h messen können.

N6 Die App muss Schüsse aus 11 Metern Entfernung messen können.

N7 Die App muss Schüsse mit einem Schusswinkel $\alpha \leq 45°$ messen können

N8 Der Messfehler darf $\pm 2$ km/h (verglichen mit einer händischen Auswertung der zum Testen genutzten Videos) nicht überschreiten.

Die Kürzel NX beziehungsweise FX werden genutzt, um diese Anforderungen im späteren Verlauf zu referenzieren. Die gestellten Anforderungen werden bei der Erstellung des Konzeptes mit eingeplant. Zudem wird während der Arbeit in regelmäßigen Abständen evaluiert, ob der aktuelle Projektfortschritt auf das Konzept hinarbeitet und das soweit entstandene Projekt die gestellten Anforderungen abdeckt. Zu diesen Anforderungen sei zu bemerken, dass sich diese auf eine fertige App beziehen von der in dieser Arbeit lediglich ein Prototyp erstellt wird. In dieser Arbeit wird auf diese App hingearbeitet, sodass diese Anforderungen beim Entwerfen und Designen der App und der Algorithmen zu berücksichtigen sind. Es können aber nicht alle Anforderungen bereits in dieser Arbeit erfüllt werden.

# 5 Betrachtung erwartbarer Abweichungen

Teil dieser Arbeit ist die theoretische Betrachtung der Abweichungen, zu denen es während der Messung kommen kann. Zunächst einmal werden einige Annahmen getroffen, die für die Berechnung der Abweichungen wichtig sind. Unter Voraussetzung einer perfekten Lokalisierung des Balles innerhalb des Bildes (es wird im Folgenden davon ausgegangen, der Ball würde von einem Algorithmus perfekt im aufgenommenen Bild detektiert werden) kann es bei der Messung zu zwei verschiedenen Arten von Abweichungen kommen. Die einen entstehen durch die seitliche Auslenkung der Flugbahn des Balles, die nicht gemessen wird[1]. Die andere durch Bewegungen des Smartphones, die während der Messungen stattfinden.

## 5.1 Vorüberlegungen

Zunächst einmal sind Vorüberlegungen anzustellen, die die Flugbahn und Geschwindigkeit des Balles betreffen.

### 5.1.1 Schussgeschwindigkeit

Wie in den Grundlagen bereits erwähnt, kann ein Fußball bis zu 120 km/h schnell werden. Die Distanz von 11 Metern vom Elfmeterpunkt bis zum Tor ist somit in maximal 0,3 Sekunden überschritten. Wenn man davon ausgeht, dass gängige Kameras und Handys Bildraten von 24 oder 30 Bildern pro Sekunde (fps) haben, vergeht zwischen 2 Bildern jeweils eine Zeit von weniger als 0,042 (24 fps) bzw. 0,034 Sekunden (30 fps). Wird ein Ball mit der Geschwindigkeit von 120 km/h vom 11 Meter Punkt aus geschossen, so legt

[1] In theoretischen Vorüberlegungen und Absprachen mit dem Auftraggeber wurde die Schwierigkeit der Messung der seitlichen Bewegung besprochen. Es wurde die Absprache getroffen, dass diese nicht mit betrachtet werden soll.

der Ball zwischen den Bildern eine Distanz von 1,389 Metern (bei 24 fps) bzw. 1,111 Metern (bei 30 fps) zurück.

$$\frac{120\frac{km}{h}}{3,6\frac{s \cdot km}{h \cdot m}} \cdot \frac{1}{30} s = 1,11m \tag{5.1}$$

Geht man davon aus, dass die Distanz von 11 Metern mit dem Handy gefilmt wird und, dass die Szene in Full HD Auflösung aufgenommen wird, dann kann abgeschätzt werden, dass der Ball von Bild zu Bild bis zu 250 Pixel zurücklegen kann.

$$1920\,pixel \cdot \frac{1,389\,m}{11\,m} = 242,4\,pixel \tag{5.2}$$

Bei kürzeren Distanzen ist diese Anzahl dementsprechend größer.

### 5.1.2 Schusswinkel

Da es um das Trainieren der Schussgeschwindigkeit geht, ist anzunehmen, dass der Ball sich tendenziell eher vorwärts bewegt, als nach oben. Damit ist gemeint, dass die Vorwärts-Komponente der Geschwindigkeit des Balles größer ist, als die aufwärts gerichtete Komponente. Folglich kann davon ausgegangenen werden, dass der Schusswinkel $\alpha$ des Balles maximal 45 Grad betragen wird. Zudem ist klar, dass sich der Ball nicht nach unten bewegen kann, was bedeutet, dass der Winkel definitiv größer ist als 0.

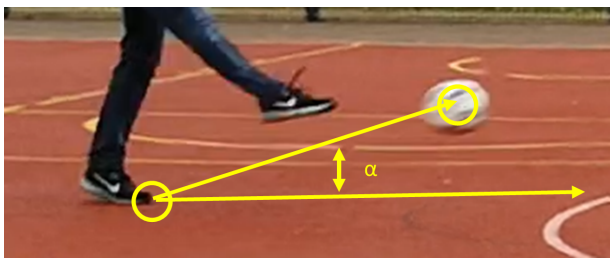


Abbildung 5.1: Vertikaler Schusswinkel

Somit kann festgehalten werden, dass für den Schusswinkel $\alpha$ gilt:

$$0 \leq \alpha \leq 45 \tag{5.3}$$

Auch die Bewegung des Balles nach rechts oder links aus Sicht des Schießenden ist wichtig. Von diesen Winkeln hängen die zu erwartenden Abweichungen bei der Messung ab. Um diese Winkel an dieser Stelle abschätzen zu können, wird davon ausgegangen, dass im Normalfall das Tor bei einem reinen Schusstraining getroffen wird. Nach der Norm DIN 7900 ist ein Fußballtor in der Profiklasse genau 7,32 Meter breit[2].

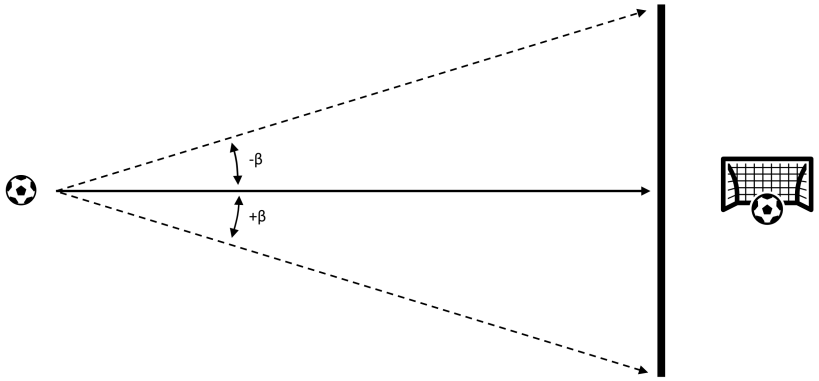


Abbildung 5.2: Maximaler Schusswinkel

Bei einer Entfernung von 11 Metern zum Tor bedeutet dies, dass der Winkel $\beta$ maximal $\pm$ 18,4 Grad betragen kann.

$$\text{atan}\left(\frac{7,32m}{2 \cdot 11m}\right) = 18,4° \tag{5.4}$$

### 5.1.3 Zu erwartende Kameraentfernung

Eine weitere Konstante, die zu beachten ist, ist die minimale Entfernung, die die Kamera beim Aufnehmen von der Schusslinie haben kann, damit die gesamte Szene auf den Sensor der Kamera passt. Um diese Entfernung abschätzen zu können, müssen der Kamerawinkel $\gamma$, aber auch die Schussdistanz bekannt sein. Da die Schussdistanz variabel sein kann, wird die Kameraentfernung als Faktor angegeben, um wie viel die Kameraentfernung anders ist, im Vergleich zur Schussdistanz.

[2]Quelle: https://de.wikipedia.org/wiki/Tor_(Fu%C3%9Fball) [Letzter Zugriff: 22.02.2021]

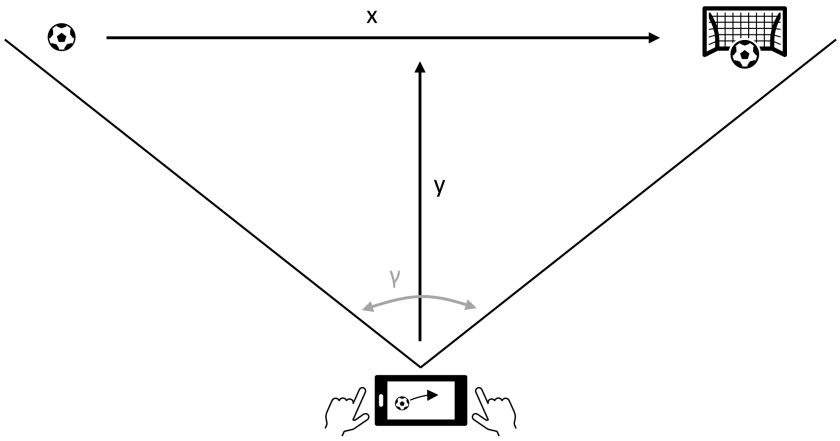


Abbildung 5.3: Kameraentfernung

Im Folgenden wird die Schussdistanz mit $x$ und die Kameraentfernung mit $y$ referenziert. Der gesuchte Faktor wird als $f$ bezeichnet, und entspricht:

$$f = \frac{y}{x} \tag{5.5}$$

Wie in Abbildung 5.3 zu sehen ist, kann die Schussdistanz aus

$$x = 2 \cdot y \cdot \tan\left(\frac{\gamma}{2}\right) \tag{5.6}$$

berechnet werden. Daraus ergibt sich für die minimale Kameradistanz, ein Faktor von:

$$f = \frac{1}{2 \cdot \tan\left(\frac{\gamma}{2}\right)} \tag{5.7}$$

Ist die Kameraentfernung geringer, als die Schussdistanz multipliziert mit diesem Faktor, so passt der Schuss nicht ganz auf den Sensor. Deswegen muss gelten:

$$y \geq x \cdot f \tag{5.8}$$

Es bleibt die Frage, wie groß der Kamerawinkel $\gamma$ ist. Die Brennweite der Hauptkameras der gängigen Smartphone-Modelle ist annähernd gleich. In einem Test mit dem für diese Arbeit zur Verfügung stehenden Modell (Huawei P20 Pro), konnte ein Bildwinkel von etwas weniger als 70° gemessen werden. Daraus ergibt sich ein Faktor von 0,714. Um die Abweichungen im Folgenden nach unten abschätzen zu können, wird von einem Faktor $f$ von minimal 0,7 ausgegangen.

## 5.2 Abweichung durch den vertikalen Schusswinkel

Zunächst geht es um die Abweichungen, die entstehen, wenn der Ball im Flug nach links oder rechts abweicht. Dabei stellt sich die gesamte Situation wie in Abbildung 5.4 gezeigt dar. Es wird die Projektion der Flugbahn auf eine imaginäre horizontale Ebene betrachtet. In dieser Abbildung seien $d_S$ die Schussdistanz und $d_K$ die Kameraentfernung.

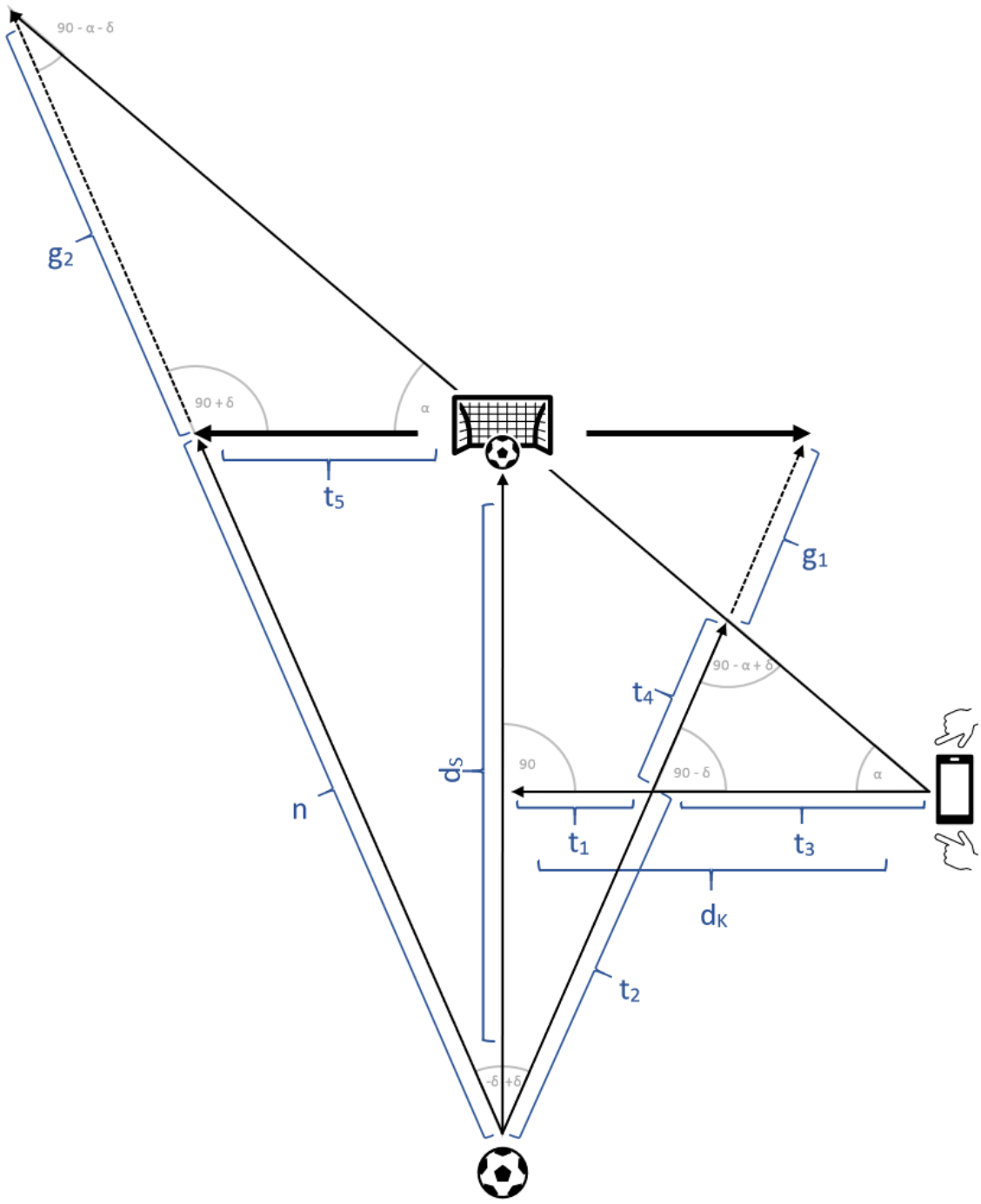


Abbildung 5.4: Flugbahnen bei seitlicher Auslenkung

In Abbildung 5.4 sei:

$$0 \leq \delta \leq \beta \tag{5.9}$$

und

$$\alpha \leq \frac{\gamma}{2} \tag{5.10}$$

Die Kameraentfernung (entspricht $t_1 + t_2$) entspricht dabei mindestens dem 0,7-fachen der Schussentfernung. Gemäß der Abbildung 5.4 berechnet sich die Länge der Schussbahn $n$, abhängig vom Winkel $\delta$:

$$n = \frac{d_S}{\cos(\delta)} \tag{5.11}$$

Dieser Wert reicht jedoch nicht aus, um die Abweichungen zu betrachten. Denn dazu muss die Sichtweise der Kamera mit eingerechnet werden, wie in Abbildung 5.4 grafisch dargestellt ist. Um so die Abweichungen berechnen zu können, sind folgende Rechenschritte nötig.

$$\alpha = \operatorname{atan}\left(2 \cdot \frac{d_K}{d_S}\right) = \operatorname{atan}\left(\frac{1}{2 \cdot f}\right) \tag{5.12}$$

Für die temporär genutzten Entfernungswerte gilt:

$$t_1 = \tan(\delta) \cdot 0,5 \cdot d_S \tag{5.13}$$

$$t_2 = 0,5 \cdot \frac{d_S}{\cos(\delta)} \tag{5.14}$$

$$t_3 = d_K - t_1 \tag{5.15}$$

$$t_4 = \sin(\alpha) \cdot \frac{t_3}{\sin(90° - \delta - \alpha)} \tag{5.16}$$

$$t_5 = tan(\delta) \cdot d_S \tag{5.17}$$

Daraus lassen sich die gesuchten Änderungen ($g_1$ und $g_2$) gegenüber der, als $n$ betrachteten Distanz, berechnen.

$$g_1 = n - t_2 - t_4 \tag{5.18}$$

$$g_2 = \sin(\alpha) \cdot \frac{t5}{\sin(90° - \alpha - \delta)} \tag{5.19}$$

Damit ergibt sich, bei Abweichung nach rechts, die Länge der Flugbahnen in Abhängigkeit des Winkels $\delta$ und des Entfernungsfaktors $f$ als:

$$l = n + g_2 \tag{5.20}$$

Für Abweichungen nach links ergibt sich analog:

$$l = n - g_1 \tag{5.21}$$

Die beiden Abbildungen 5.5 und 5.6 geben eine Übersicht über die zu erwartenden Abweichungen. Die linke Abbildung zeigt die Länge der wirklichen Flugbahn im Vergleich zu der von der Kamera gedachten Länge also:

$$\frac{\text{wirkliche Flugbahn}}{\text{gedachte Flugbahn}} = \frac{l}{d_S} \tag{5.22}$$

Die rechte Abbildung zeigt die sich daraus ergebende Abweichung der gemessenen Schussgeschwindigkeit im Vergleich zur Wirklichen. Diese Abweichung entspricht der Abweichung in der gemessenen Geschwindigkeit.

$$\text{Abweichung} = 100\% \cdot \left(\frac{l}{d_S} - 1\right) \tag{5.23}$$

Die Bilder zeigen die Abweichungen für einen Winkel von $-18,5° \leq \delta \leq 18,5°$ und einem Faktor von $0,7 \leq f \leq 1,5$

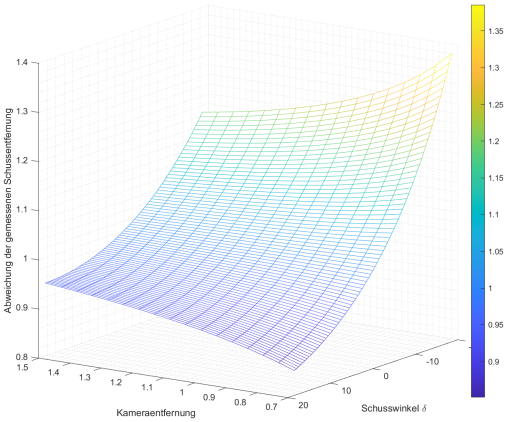


Abbildung 5.5: Unterschiede in der Länge der Flugbahn

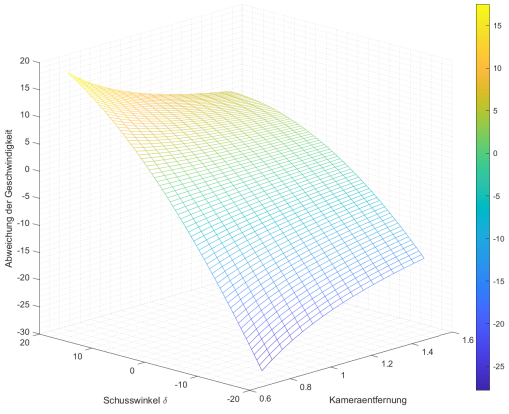


Abbildung 5.6: Abweichungen der gemessenen Geschwindigkeit

Die Berechnungen zeigen, dass Abweichungen zwischen $-28\%$ und $+18\%$ entstehen können. Allerdings wurde die Abweichung mit $\pm 18,5°$ sehr großzügig abgeschätzt. Bei einem Schusskrafttraining kann davon ausgegangen werden, dass der Spieler das Tor mittig trifft. Dementsprechend wären die Abweichungen um einiges geringer. Beim Erstellen der Videos zur Vorbereitung der Entwicklung der Algorithmen sind nur Abweichungen in einem Bereich deutlich unter 10 Grad aufgetreten. Geht man also davon aus, dass der Winkel maximal 10 Grad betragen kann und die Kameraentfernung mindestens den Faktor 1 hat, so liegen die möglichen Abweichungen nur noch zwischen +7 und -10 Prozent.

## 5.3 Abweichungen auf Grund von Kamerabewegungen

An dieser Stelle werden die Abweichungen betrachtet die entstehen, wenn sich das Smartphone, mit dem gefilmt wird, bewegt. Die Bewegungen lassen sich dabei in zwei Komponenten aufteilen, die hier zur Vereinfachung getrennt voneinander betrachtet werden. Zum einen gibt es die horizontale Bewegung, bei der sich das Smartphone mit fixer vertikaler Position bewegt. Zum anderen gibt es die genau gegenteilige Bewegung, bei der bei fixer horizontaler Position, die Bewegung in vertikaler Richtung stattfindet. Beide

Bewegungen lassen sich in je zwei weitere Kategorien aufteilen. Auf der einen Seite die parallelen Bewegungen, bei denen das Smartphone mit oder entgegen der Schussrichtung bewegt wird. Auf der anderen Seite die Kippbewegungen, bei denen das Smartphone um die jeweilige Achse gekippt wird. Die Abbildung 5.7 verdeutlicht diese Bewegungen.

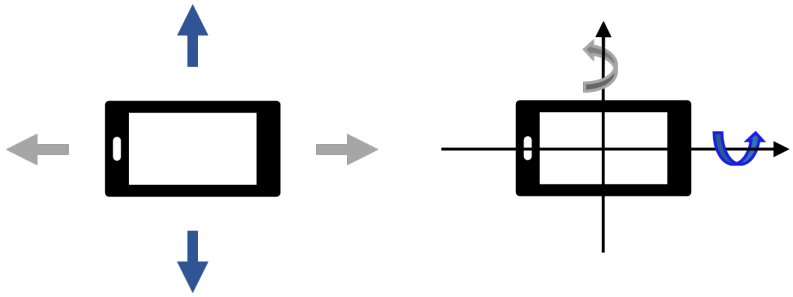

Abbildung 5.7: Verschiedene Bewegungen

Blau steht dabei für die Bewegungen mit vertikalem Einfluss und grau für Bewegungen mit horizontalem Einfluss. Bei den Kippbewegungen wird das Smartphone dabei minimal um die jeweils andere Achse rotiert. Bei einer vertikalen Kippbewegung wird das Smartphone um die horizontale Achse rotiert. Im Folgenden werden die Auswirkungen der einzelnen Bewegungen genauer betrachtet.

### 5.3.1 Messung von Kippbewegungen

Bevor jedoch auf die Abweichungen eingegangen wird, ist zu betrachten, wie groß diese Kippwinkel eigentlich werden können. Um einen Überblick zu bekommen, mit was für Kippwinkel gerechnet werden muss, wird ein Test durchgeführt. Bei diesem Test wird ein Tor aus 10 Metern Entfernung gefilmt. Anschließend wird das Video ausgewertet, indem die Pixel-Koordinaten der oberen linken Ecke des Tores per Hand ausgelesen werden. Aus diesen Koordinaten und dem Bildwinkel des Smartphones kann der Winkel berechnet werden, um den das Smartphone gekippt worden sein muss. Dabei wird angenommen, dass die auftretende Bewegung eine reine Kippbewegung ist (dadurch sind die gemessenen Werte größer gleich den wirklichen aufgetretenen Werten). Das Ergebnis ist in Abbildung 5.8 festgehalten.

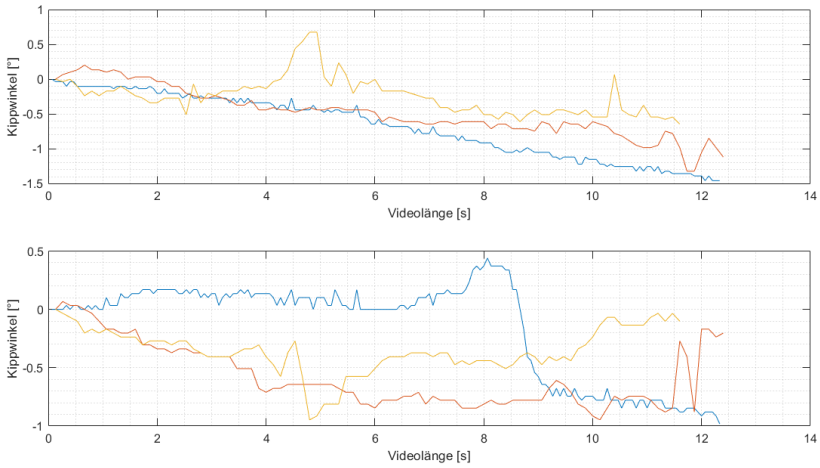


Abbildung 5.8: Messung Kippwinkel

Es wurden insgesamt drei Videos aufgenommen. Das erste Video (blau) wurde mit der internen Kamera-App aufgenommen und verfügt so über eine Videostabilisierung. Die andern beiden Videos (rot und gelb) wurden mit einem Prototyp der App aufgezeichnet und verfügen über keine Videostabilisierung. Die Videos wurden aus der Hand an einem relative kalten Wintertag, bei leichtem Wind aufgenommen, sie bieten also eine gute Abschätzung, wie sehr das Smartphone kippen kann. In den Verläufen ist zu sehen, dass der Kippwinkel um bis zu $1,5°$ variieren kann. Als Abschätzung wird im Folgenden daher ein Winkel von weniger als $2°$ betrachtet.

### 5.3.2 Horizontale Kamerabewegung

Durch die horizontale Bewegung kann die benötigte Schussweite, die gebraucht wird, um das im Smartphone gedachte Ziel zu erreichen, erhöht oder vermindert werden. Da sich aber nur die gemessene, nicht aber die wirkliche Schussweite ändert, ist die benötigte Zeit zum Erreichen des Ziels aus Sicht des Smartphones abweichend. Daraus resultieren entsprechende Abweichungen, in der gemessenen Geschwindigkeit. Dies ist beispielhaft in Abbildung 5.9 gezeigt. Dabei ist eine verschiebende Bewegung (links) und eine kippende Bewegung (rechts) gezeigt, bei der der Ball eine längere Strecke zurücklegen muss, um das Ziel für das Smartphone zu erreichen. Die gemessene Geschwindigkeit wäre also

langsamer als sie eigentlich ist. Der rote Pfeil symbolisiert das im Smartphone markierte Ziel. Zu sehen ist, wie dieses durch die Bewegungen nach hinten verschoben wird.

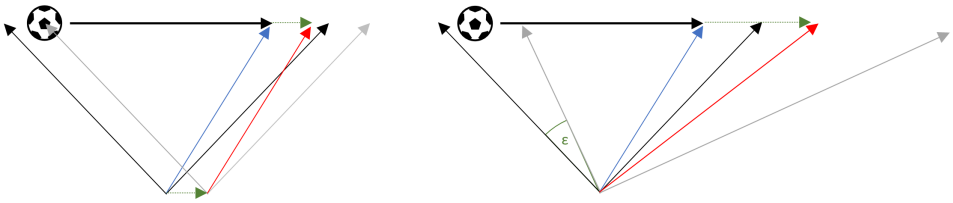


Abbildung 5.9: Verschiebende Bewegungen

Durch die links zu sehende verschiebende Bewegung wird die Strecke, die der Ball zurückgelegt hat, um dieselbe Strecke (grüner Pfeil nach der Ballflugbahn) gedehnt, wie die Bewegung selbst (grüner Pfeil unten im Bild). Die Abweichung $\Delta t$, die in diesem Fall entsteht, lässt sich wie folgt berechnen. Der durch den grünen Pfeil markierte Streckenunterschied wird als $\Delta x$ bezeichnet.

$$\Delta t = 1 - \left( \frac{d_S + \Delta x}{d_S} \right) \cdot 100 \tag{5.24}$$

Um mit dieser Berechnung die Abweichungen abschätzen zu können, wird angenommen, dass sich der Ball ausschließlich vorwärts bewegt, ohne zusätzliche Höhenkomponente. Würde die Bewegung des Balles zusätzlich über letztere verfügen, so wäre diese nicht von der Bewegung (des Smartphones) betroffen, würde aber trotzdem zur Ballgeschwindigkeit beitragen. Die Abweichungen wären also anteilsmäßig geringer. Die hier berechnete Abweichung ist also als eine maximale Abschätzung zu betrachten. Für die in Abbildung 5.10 gezeigten Abweichungen wurde zusätzlich von einer Schussentfernung von 6 Metern ausgegangen, wodurch kleinere Bewegungen einen größeren Einfluss haben, als bei weiteren Entfernungen. Die so im Allgemeinen zu erwartenden Abweichungen sind also kleiner gleich denen in dieser Abbildung dargestellten.

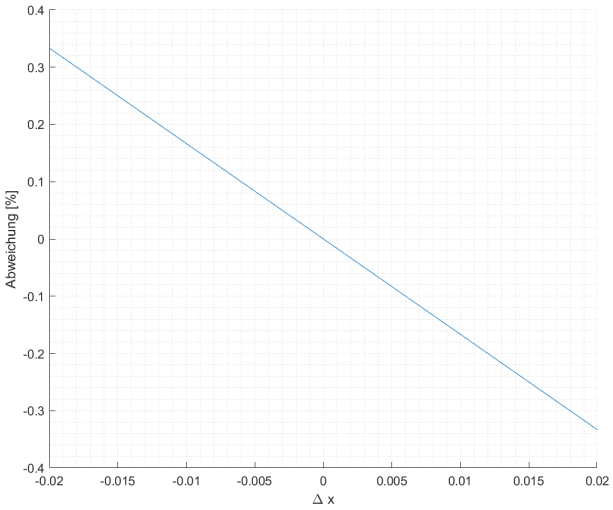


Abbildung 5.10: Abweichungen bei verschiebenden Bewegungen

Es ergeben sich dabei Abweichungen von lediglich etwas mehr als ±0.3 Prozent. Diese jedoch bei einer Bewegung von 2 Zentimetern, was als extrem viel einzustufen ist. Die Abweichungen der verschiebenden Bewegung können also durchaus vernachlässigt werden. Im Gegensatz dazu ist das bei den Kippbewegungen nicht der Fall. Dort können bereits bei kleinen Änderungen im Winkel relativ große Abweichungen entstehen. Um die Berechnungen aufzustellen wird Abbildung 5.11 betrachtet.

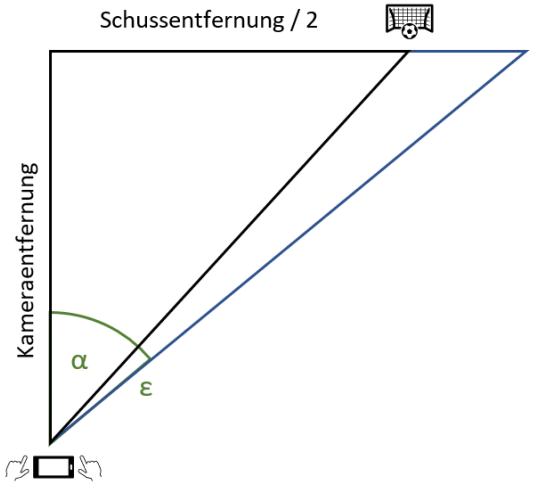


Abbildung 5.11: Kippbewegung

In dem vom Winkel $\alpha$ beschriebenen Dreieck findet normalerweise der Schuss statt. Durch den Kippwinkel $\epsilon$ ändert sich dieses und es kommt zu Abweichungen in den Messungen. Diese hängen vor allem von dem Verhältnis der Schussentfernung zur Kameraentfernung ab. Der Faktor $n$, um den sich die Gegenkathete im gezeigten Bild in Abhängigkeit der Winkel $\epsilon$ und $\alpha$ ändert, ist gegeben durch:

$$n = \frac{tan(\alpha + \epsilon)}{\tan(\alpha)} \tag{5.25}$$

Zunächst wird an dieser Stelle betrachtet, wie sich die Abweichungen für einen variablen Winkel $\alpha$ ändern, wenn von einer konstanten Verwacklung $\epsilon$ ausgegangen wird. Dabei wird für $\epsilon$ zunächst einmal ein Wert von 1° angenommen, sowie ein $f$ von 0,7 bis 1,5. Die gesamte Abweichung, der gemessenen Schussgeschwindigkeit gegenüber der realen, lässt sich berechnen durch:

$$n = \left(1 - \frac{\tan(\alpha) + \tan(\alpha + \epsilon)}{2 \cdot \tan(\alpha)}\right) \cdot 100 \tag{5.26}$$

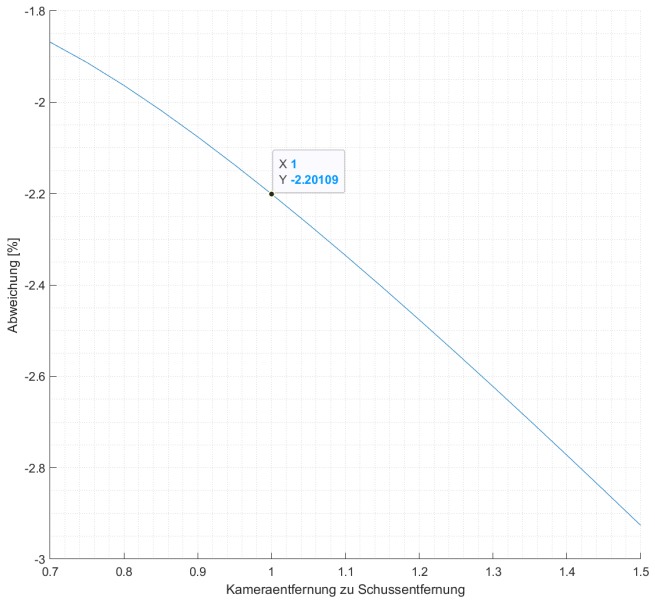


Abbildung 5.12: Abweichungen bei Kippbewegungen

Aus Abbildung 5.12 ist zu erkennen, dass die Abweichungen umso größer werden je weiter die filmende Person vom Geschehen entfernt ist. Dies steht im Widerspruch zu den in Kapitel 5.2 erlangten Erkenntnis, dass die Abweichungen dann geringer sind, wenn die Kameraentfernung größer ist. Aus diesem Grund muss ein Kompromiss zwischen diesen beiden möglichen Abweichungen gefunden werden. Im Folgenden gilt daher, dass der Entfernungsfaktor $f$ von der Schussentfernung zur Kameraentfernung 1 beträgt. DAbbildung 5.13 zeigt die Abweichungen bei einem variablen Winkel $\epsilon$ im Bereich von $\pm 2°$.

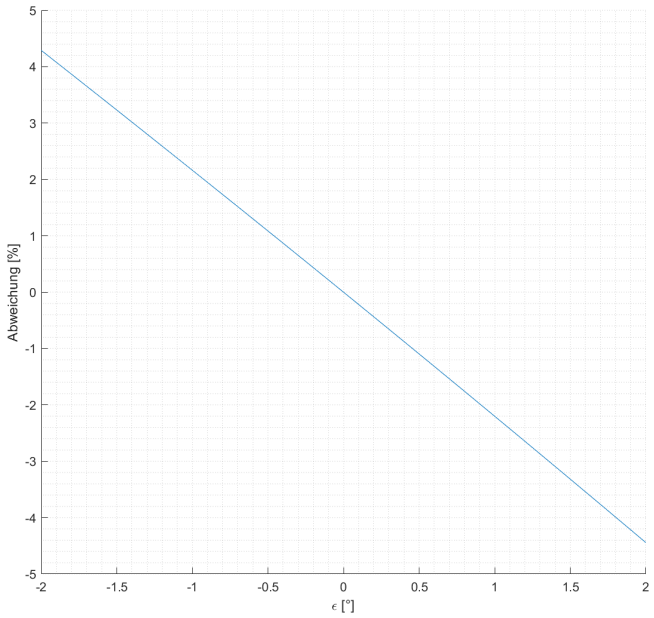


Abbildung 5.13: Abweichungen bei verschiedenen Kippwinkeln

Die Abbildungen zeigen, dass bei verschiebenden Bewegungen kaum Abweichungen entstehen, sodass diese Bewegung bei den Messungen vernachlässigt werden können. Anders ist es bei kippenden Bewegungen. Dort entstehen bei Bewegungen von bis zu 2 Grad Abweichungen von $\pm 4,5\%$.

### 5.3.3 Vertikale Kamerabewegung

An dieser Stelle werden die Abweichungen betrachtet, die entstehen, wenn sich das Smartphone vertikal bewegt. Dabei werden die verschiebenden Bewegungen nicht weiter berücksichtigt, da sie lediglich zu minimalen Abweichungen führen. Bei vertikalen Abweichungen

werden an dieser Stelle die verschiedenen Schusswinkel $\alpha$ von 0 bis 45 Grad untersucht. Es werden dabei lediglich die beiden Extreme angenommen, bei denen das Smartphone um 2° nach oben oder unten kippt. Der Entfernungsfaktor beträgt 1. Zudem wird angenommen, dass die Kippbewegung gleichmäßig über den gesamten Schuss verteilt erfolgt. So kann die Abweichung durch eine Veränderung des Schusswinkels $\alpha$ um $\epsilon$ betrachtet werden.

$$\text{Abweichung} = \left( \frac{cos(\alpha + \epsilon)}{cos(\alpha)} - 1 \right) \cdot 100 \tag{5.27}$$

Die Abbildungen 5.14 und 5.15 zeigen diese Abweichungen in Abhängigkeit des Schusswinkels $\alpha$. Links zu sehen ist die Abweichung bei einer Bewegung gegen die Schussrichtung, rechts die Abweichung einer Bewegung mit der Schussrichtung.

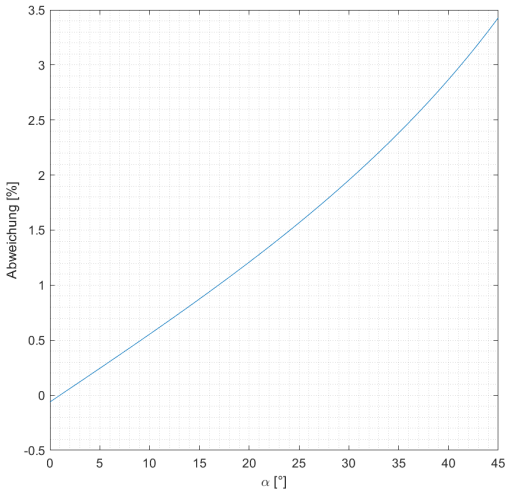


Abbildung 5.14: Vertikale Kippbewegung mit Schussrichtung

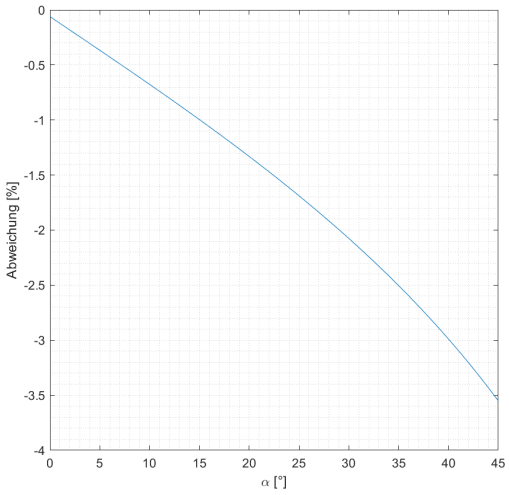


Abbildung 5.15: Vertikale Kippbewegung gegen Schussrichtung

Betrachtet man die Möglichkeit, dass bei einem parabelförmigen Schussverlauf die Abweichung zweimal stattfinden kann, so sind die maximalen Abweichungen doppelt so groß, wie die, die in den Abbildungen 5.14 und 5.15 gezeigt sind. So können im schlimmsten Fall also Abweichungen von bis zu ±7% entstehen. Auch bei diesen Abweichungen ist in Betracht zu ziehen, dass diese zu kompensieren sind.

## 5.4 Zusammenfassung

Es gibt also eine Reihe von Abweichungen, die zu betrachten sind. Fasst man alle Abweichungen zusammen, kann es sein, dass die Messung eine Abweichung von etwa ±20% hat. Dementsprechend würde ein Schuss, der eine Geschwindigkeit von 20 $\frac{m}{s}$ aufweist, als Messwert einen Wert zwischen 16 und 24 $\frac{m}{s}$ aufweisen. Diese Abweichungen sind relativ groß. Es handelt sich allerdings um grobe Abschätzungen. Die wirklichen Abweichungen werden um einiges kleiner ausfallen. Beispielsweise ist bei einem Schuss, der etwa eine Sekunde dauert nicht davon auszugehen, dass eine Kippung um 2° stattfindet. Diese Abweichung hat in den getesteten Videos über die Videolänge von etwa 12 Sekunden stattgefunden. Zudem ist fraglich, ob der Schuss bei einer plötzlichen Verwacklung, um die besagten 2 Grad, überhaupt verfolgt werden kann. Die realen Abweichungen werden um einiges geringer ausfallen, als die hier theoretisch betrachteten. Es ist dementsprechend festzuhalten, dass von einer ziemlich guten Genauigkeit ausgegangen werden kann.

# 6 Konzept

Es gibt zwei große Betriebssysteme für Smartphones, für die Apps programmiert werden können; Android und iOS. Diese haben zusammen ein Marktanteil von knapp 100 Prozent[1]. Um die Anforderung des Auftraggebers (F2) zu erfüllen, wird die App in Android programmiert. Dies deckt sich auch mit den Anforderungen des Entwicklers, die Umsetzung durch möglichst bekannte Plattformen vorzunehmen. Aus diesem Grund wird die Umsetzung in Android-Studio und mit der Programmiersprache Java erfolgen. Mit dieser Plattform und dieser Programmiersprache können die geforderten Funktionen und Anforderungen umgesetzt werden und es sind bereits Erfahrungen im Umgang mit diesen vorhanden. Dass die Umsetzung in Android stattfindet, hat einen weiteren Vorteil. Android hat in Deutschland mit 60 bis 70 Prozent einen höheren Marktanteil als iOS, die App wird demnach für mehr Nutzer zur Verfügung stehen.

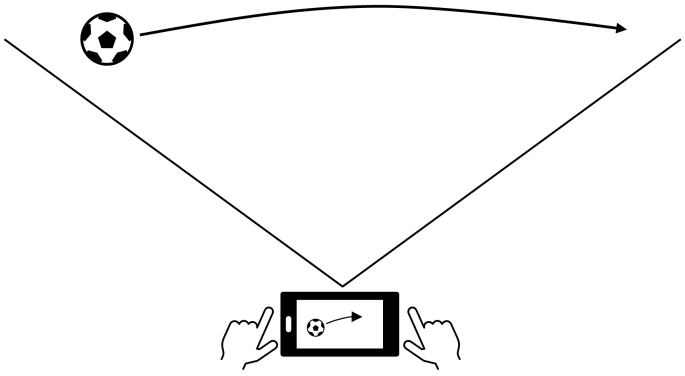

Abbildung 6.1: Nutzungskontext der App

[1] Quelle: https://de.statista.com/statistik/daten/studie/184332/umfrage/marktanteil-der-mobilen-betriebssysteme-in-deutschland-seit-2009/#professional [Letzter Zugriff: 01.03.2021]

Mit dem Smartphone und der App wird der Schuss oder Wurf von der Seite, etwa mittig der Flugbahn, gefilmt. Dabei kann es zwar zu den in Kapitel 5.2 beschriebenen Abweichungen kommen, was den Anforderungen der professionelleren Nutzer widersprechen könnte, doch ist bei diesen Nutzern auch zu erwarten, dass die vertikalen Abweichungen des Schusses gering ausfallen. Um aus dieser Position die Geschwindigkeit berechnen zu können, müssen entweder die Flugdistanz oder die Distanz zur Flugbahn und der Kamerawinkel bekannt sein. Da es beim Fußball einen festen Anfangs- und Endpunkt gibt wird hier die einfachere Methode genutzt, sodass mit der Flugdistanz nur ein Wert einzugeben ist. Bei der Messung von Bällen, die zum Beispiel geworfen werden, ist es nicht immer einfach die Entfernung, die der Ball fliegt vorherzusagen. Aus diesem Grund ist es in diesem Fall einfacher, wenn die Distanz zur Flugbahn und der Kamerawinkel eingegeben werden.
Im Folgenden wird auf die Konzepte und die Designs der Bildverarbeitung und der App genauer eingegangen.

## 6.1 Konzept und Design der Bildverarbeitung

Die Bildverarbeitung wird am Computer entworfen. Zum Testen und Entwickeln werden Videos aufgenommen, die dann am Computer ausgewertet werden. Dazu wird die Bibliothek OpenCV verwendet. Diese verfügt zum einen über alle benötigten Funktionen und zum anderen sind bereits einige Erfahrungen mit dieser Bibliothek vorhanden. OpenCV kann in der Desktop-Entwicklung in C++ und Python genutzt und später in Android implementiert werden. Um die Bildverarbeitung in Android implementieren zu können, müssen die Funktionen in C++ geschrieben werden. In dieser Arbeit wird jedoch nur die Bildverarbeitung an sich und eine App als Prototyp ohne notwendige Implementierung der Bildverarbeitung geschaffen. Aus diesem Grund ist es nicht zwingen notwendig die Bildverarbeitung in C++ zu entwerfen. Um die Anforderung des Entwicklers zu erfüllen, mit bekannten Plattformen zu arbeiten, wird die Entwicklung zunächst in Python stattfinden. Für etwaige Weiterentwickler ist es dann notwendig die Funktionen in C++ zu übersetzen.
Um der Anforderung F1 gerecht zu werden, wird die Bildverarbeitung in zwei Teile aufgeteilt. Dies wird gemacht, um den Vorteil beim Fußball zu nutzen, dass vor der Messung die Position fest und bekannt ist[2].
Es wird an dieser Stelle zunächst auf die Bildverarbeitung eingegangen, mit der die

[2]Wenn zu Beginn des Schusses der Ball fest positioniert auf dem Boden liegt.

Geschwindigkeiten aller Bälle gemessen werden kann. Dieser wird im Folgenden als allgemeiner Algorithmus bezeichnet. Anschließend wird das Konzept für den, für Fußball optimierten Algorithmus vorgestellt, welcher folgend als optimierter Algorithmus bezeichnet wird.

Es gibt mehrere Möglichkeiten die Detektion des Balles vorzunehmen, insbesondere:

- Detektion durch Farbfilterung
- Detektion durch Kreistransformation
- Detektion bei Bewegung durch Differenzbilder
- Detektion durch Objekt Detektion mit einem Neuronalen Netzwerk

Von der Farbfilterung ist abzusehen, da es Bälle in unzählig verschiedenen Farbvarianten gibt. Zwar könnte der Nutzer die Farbe vorher eingeben, doch besteht dann immer die Gefahr, dass sich diese Farben auch im Hintergrund befinden, sodass es zu falschen Detektionen kommt. Ein ähnliches Problem ist auch bei der Kreistransformation zu befürchten. Es kann auch da nicht mit Sicherheit gesagt werden, dass sich im Hintergrund keine anderen kreisförmigen Objekte befinden. Darüber hinaus kann es bei schnellen Ballbewegungen dazu kommen, dass der Ball verwischt ist, sodass dieser keinen erkennbaren Kreis mehr darstellt. Bei ersten Tests der Kreistransformation (mit und ohne Kombination mit Differenzbilder), hat sich zudem gezeigt, dass diese nicht für den geplanten Einsatz geeignet ist. Es werden willkürlich wirkend Kreise überall im Bild erkannt. Als nächste Option bietet sich die Objekt Detektion mit einem neuronalen Netzwerk an. Auf der einen Seite ist diese Methode sehr gut geeignet, um den Ball im Bild zu erkennen. Auf der anderen Seite stellt das Trainieren eines neuronalen Netzes eine große Hürde dar. Es würden eine Menge Daten benötigt werden; Videomaterial von den verschiedensten Bällen, auf den verschiedensten Sportplätzen mit und ohne Menschen im Hintergrund. Die Erstellung all dieser Daten stellt im Zuge dieser Arbeit einen zu großen Zeitaufwand dar. Erschwerend hinzu kommt die aktuelle Lage mit der Covid-19-Pandemie, die die Erstellung dieser Daten kaum möglich macht. Deswegen steht dieser Möglichkeit im Kontext dieser Arbeit zu viel im Weg, weswegen eine andere Option zu wählen ist. Die Detektion der Ballpositionen im Bild wird durch die Differenzbilder vorgenommen. Diese eignen sich, im Kontext dieser Arbeit, am besten für diese Aufgabe, da durch die Differenzbilder der Fokus direkt auf die Bewegung gelegt wird, um die es bei dieser Arbeit geht. Die Bälle sind im Flug klar und deutlich zu erkennen, sodass diese schnell detektiert werden

können. Die Videos, mit denen die Bildverarbeitung arbeiten wird, werden über die Full-HD-Auflösung verfügen. Diese ist hochauflösend, sodass mit vielen Bild-Informationen gearbeitet werden kann. Zudem ist dies die momentane Standardauflösung der Videos in Kameras und Smartphones.

An dieser Stelle wird auf das Design der Bildverarbeitung betrachtet. Zunächst wird auf das Design der allgemeinen Bildverarbeitung eingegangen. Zur Detektion wird von einem Teil des Bildausschnittes die Differenz gebildet, um in diesem Bereich befindlichen Bewegungen zu detektieren, wie in Abbildung 6.2 dargestellt. Es wird nur ein Teil des Bildes verwendet, um so zu verhindern, dass andere Bewegungen von links oder rechts ins Bild kommen. Der Nutzer soll den Spieler, der den Ball wirft oder schießt, noch im Bild sehen ohne, dass dieser fälschlicherweise als Bewegung detektiert wird.

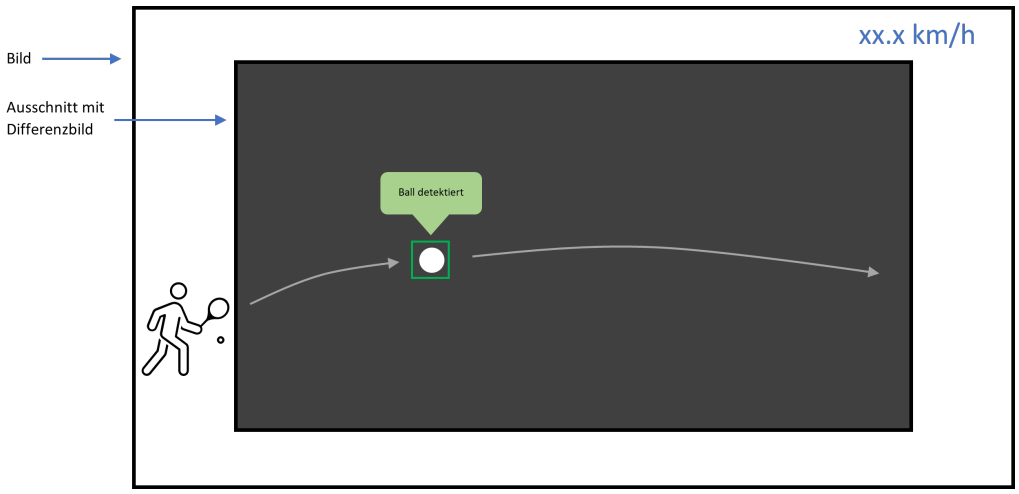


Abbildung 6.2: Konzept der allgemeinen Balldetektion

Innerhalb dieses Bereiches wird für jedes Bild die Bewegung detektiert. Um die Anforderung F5 einzubinden, erfolgt danach eine Überprüfung, ob die detektierte Bewegung ein Ball war oder Umrisse aus dem Hintergrund, die aus der Verwacklung des Smartphones resultieren. Wie diese Überprüfungen genau funktionieren, ist bei der Implementierung experimentell herauszufinden.
Da es sich bei Aktivierungen im Differenzbild, die durch Verwacklungen resultieren, vor allem um Linien durch die Umrisse von Objekten, und bei Bällen um kreisförmige Bereiche handelt, sind folgende Überprüfungen denkbar:

- Überprüfung, ob es sich bei der Aktivierung um einen Kreis handelt.
- Schwellwert, den ein gewisser Bereich um den detektierten Mittelpunkt überschreiten muss.
- Plausibilitätsüberprüfung der Position, wenn bereits mehrere Positionen vorhanden sind.

Das Resultat wird nach der Skizze in Abbildung 6.2 entwickelt. Da die Detektion nur in einem Teil des Bildes erfolgt, besteht die Möglichkeit, Verwacklungen des Smartphones softwaretechnisch zu kompensieren. Mit dieser Methode könnten Verwacklungen nachträglich reduziert werden und es könnten so auch die Abweichungen durch die Bewegung des Smartphones (vgl. 5.3) kompensiert werden. Diese Möglichkeit wird daher in Betracht gezogen, wird aber aus zeitlichen Gründen nicht vorrangig implementiert.

Auch für den optimierten Algorithmus werden die Differenzbilder verwendet, jedoch nicht ausschließlich. Wie bereits erwähnt, wird der beim Fußball bestehende Vorteil, dass zu Anfang des Schusses die Position des Balles bekannt ist, genutzt. Dieser Vorteil wird genutzt, indem die Verfolgung der Ballpositionen durch die Differenzbilder erst nach Beginn des Schusses stattfindet. Dazu muss der Nutzer die Ballposition auf dem Bildschirm markieren. Wenn der Ball in Ruhe liegt, so findet in diesem markierten Bereich keine Bewegung statt. Wenn der Ball geschossen wird, dann tritt in diesem Bild eine starke Aktivierung auf. Dies wird als Erkennungssignal genutzt, um den Beginn des Schusses zu detektieren. Von da an wird der Ball durch eine Bewegungserkennung mit Differenzbildern verfolgt. Dabei erfolgt die Detektion der Ballpositionen nicht, wie bei der allgemeinen Erkennung durch ein großes Fenster, sondern durch kleine Fenster, die aufgrund der vorherigen Positionen berechnet werden. Zu Anfang des Schusses, wenn Geschwindigkeit und Bewegungsrichtung des Balles noch unbekannt sind, sind die Fenster entsprechend größer zu wählen. Im Laufe des Schusses wird zur Ausrichtung des Suchfensters Geschwindigkeit und Bewegungsrichtung mit einbezogen werden.

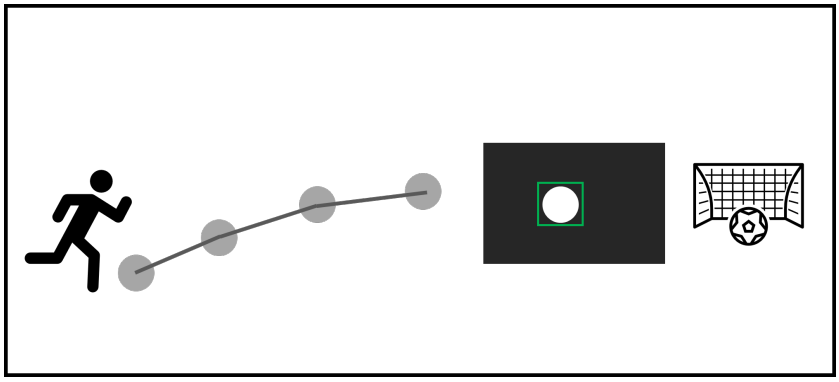

Abbildung 6.3: Detektionsfenster allgemeiner Algorithmus

Der Schuss wird für beendet erklärt, wenn der Ball das Tor erreicht hat. Dazu müssen die Positionen der beiden Torpfosten bekannt sein. Tests haben gezeigt, dass Linienfilter oder die Linientransformation die Torpfosten nicht zuverlässig erkennen können. Die Koordinaten dieser müssen also vom Nutzer eingegeben werden. Der Ablauf der Bilderkennung wird konzeptionell wie folgt aussehen.

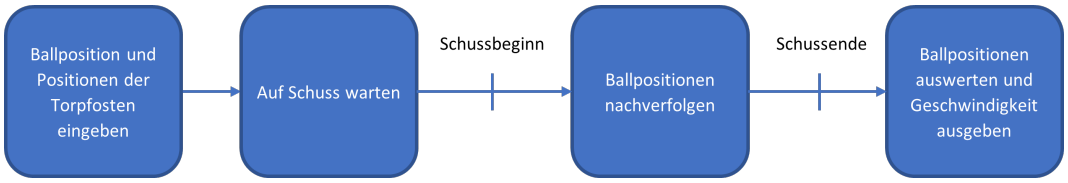


Abbildung 6.4: Konzept der für Fußball optimierten Balldetektion

Um später in die App eingefügt werden zu können, wird die Bildverarbeitung in auslagerbaren Funktionen programmiert. Um die Entfernung, die vom Nutzer einzugeben ist, in den Bilddaten wiederzufinden, sind zusätzlich beide Torpfosten vom Nutzer zu markieren. Die eingegebene Schussdistanz in Metern entspricht dann der Entfernung in den Bilddaten, die in Pixeln gemessen werden.
Aus den detektierten Ballpositionen könnte mit der, in Kapitel 2.3 beschriebenen Formel, die Parameter der Flugbahn bestimmt werden. Doch stellt sich das aufgrund der Komplexität der Formel als relativ kompliziert heraus. Eine nummerische Berechnung durch die Auswertung zwischen den einzelnen Ballpositionen ist daher zu bevorzugen. Die zu erwartenden Abweichungen durch die nummerische Auswertung sind ohnehin als sehr gering einzuschätzen.

## 6.2 Konzept und Design der Android-App

Wie in den Grundlagen beschrieben ist es mit Android-Studio möglich nativen C++ Code in das Projekt einzufügen und in der App auszuführen. Über diesen ist es möglich auch OpenCV-Funktionen zu nutzen. Dies werden genutzt, um die Bildverarbeitung in der App zu implementieren.
Das Kamerainterface (siehe Kapitel 2.1.1), das für die Prototyp-App verwendet wird, ist das OpenCV-Interface. Da das Camera2-Interface keinen Zugriff auf die Bilddaten bietet, ist es für diese Aufgabe gänzlich ungeeignet. Über das CameraX-Interface sind Beispiele und Erklärungen nahezu ausschließlich in Kotlin verfügbar. Zwar ist es allem Anschein nach besser für diese Aufgabe geeignet, um auf die Anforderung des Entwicklers einzugehen, wird es dennoch nicht für die Entwicklung des Prototyps verwendet.

Die App muss nach Anforderung N10 möglichst einfach zu bedienen sein. Aus diesem Grund werden in die App keine Activities eingefügt, die nicht direkt der Erfüllung der Anforderungen dienen. Dementsprechend wird es nach Anforderung F1 und dem eben besprochenen Konzept zur Bildverarbeitung zwei Activities zur Geschwindigkeitsmessung geben. Auch wird die App, nach Anforderung F4 und F7, über eine Spielerverwaltung verfügen. Die Activities zur Bildverarbeitung werden, um die Anforderung F3 zu erfüllen, nach Messung der Geschwindigkeit, diese anzeigen. Zudem werden sie die Möglichkeit bieten die Geschwindigkeit einem Spieler aus der Spielerverwaltung hinzuzufügen. Ist die Messung fehlerhaft, so wird gemäß F6 eine Fehlermeldung ausgegeben. Teil der Anforderung N10 ist, dass der Nutzer schnell und intuitiv versteht, wie die App zu nutzen ist. Es wird also pro Bildverarbeitungs-Activity jeweils eine Activiy zur Erklärung dieser aufrufbar sein. Dies ist nur für den Fall gedacht, dass der Nutzer trotz intuitivem Design der App noch Fragen zur Nutzung hat. Die Struktur der Activities ist Abbildung 6.5 zu entnehmen.

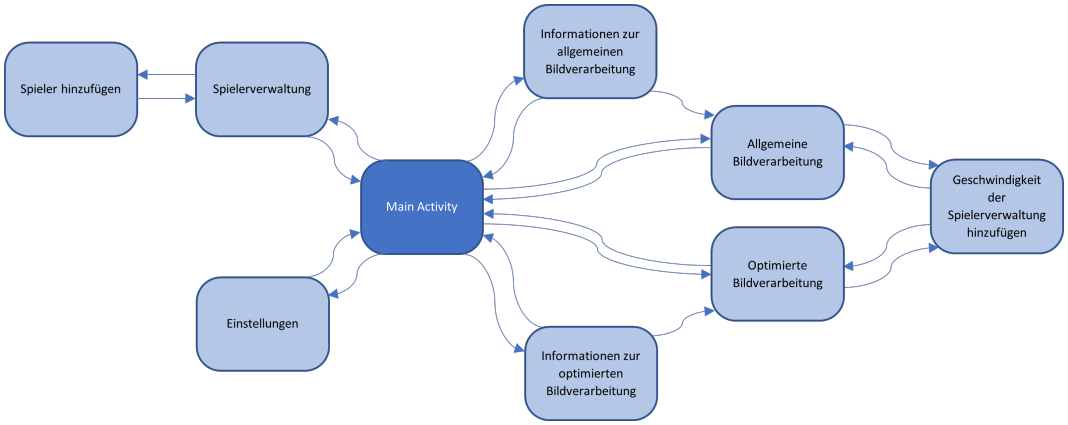


Abbildung 6.5: Navigation zwischen den Activities

Im Folgenden wird das genaue Design der App beschrieben. Abbildungen 6.6 zeigt die Designs der einzelnen Activities. Zu sehen sind die Main Activity, die Activity für die Erklärung zur Nutzungsweise und die Spielerverwaltung.

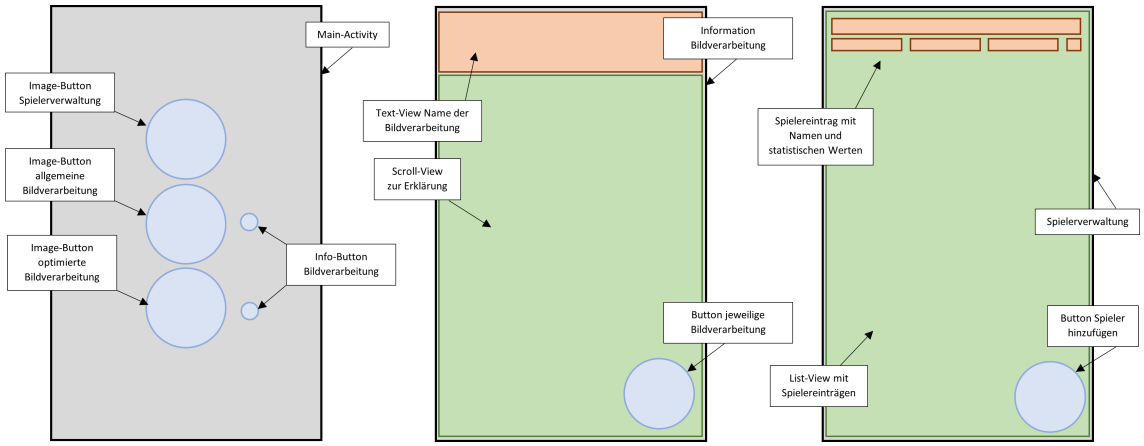


Abbildung 6.6: Design der Activities

Die Main Activity wird mit drei Buttons implementiert, die zur Spielerverwaltung und den Activities der Bildverarbeitung führen. Um Anforderung N4 zu erfüllen, werden diese als Image-Buttons ausgeführt. Die Activity für die Nutzungsanleitung verfügt über ein Scroll-View, in dem die Erklärung steht. Ein Floating-Action-Button wird die Möglichkeit zur Weiterleitung zur Bildverarbeitung bieten. Die Spielerverwaltung besteht aus

einem List-View, in dem die Spieler aufgeführt sind. Zu diesen werden statistische Werte wie Rekordwert, Mittelwert und/oder Median und/oder negativ Rekord angezeigt. Über einen Floating-Action-Button können neue Spieler hinzugefügt werden. Die Spielereinträge der Liste können nach unterschiedlichen Kriterien sortiert werden, etwa den Tagesbestwert (F7). Die Activities für die Bildverarbeitung werden wie in Abbildung 6.7 designt.

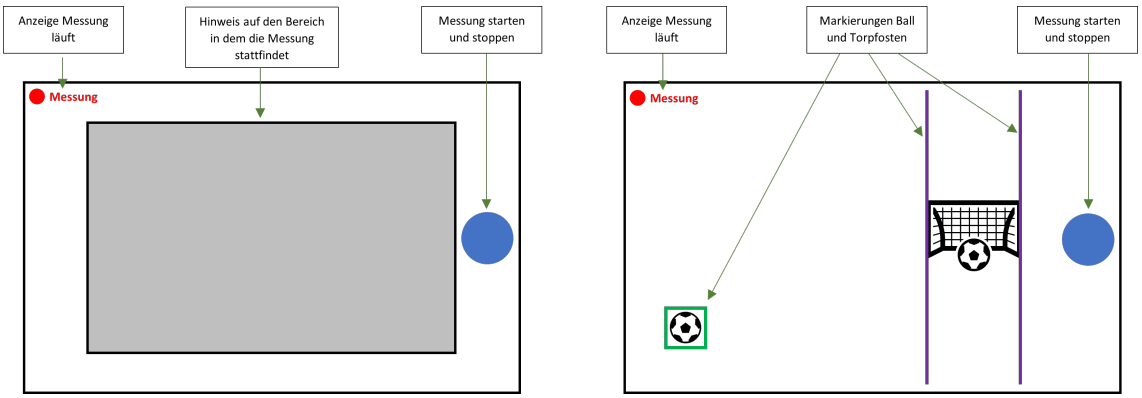


Abbildung 6.7: Design der Activities der Bildverarbeitung

Links zu sehen ist das Design der allgemeinen Bildverarbeitung, rechts das der optimierten. Beide verfügen über einen Button, mit dem die Messung gestartet und gestoppt werden kann. Je nachdem, ob eine Echtzeitimplementierung[3] möglich ist, wird diese damit gestartet oder es werden erst die Bilder aufgezeichnet und mit Stoppen der Aufzeichnung wird die Messung begonnen (siehe Abbildung 6.8). Bei der allgemeinen Bildverarbeitung wird dem Nutzer der Bereich angezeigt, in dem die Messung stattfindet. Bei der optimierten hat der Nutzer, wie im Bild zu sehen, den Ball und das Tor zu markieren.

[3]Im Kontext dieser Arbeit wird, in Bezug auf die Algorithmen, Folgendes als Echtzeit verstanden: Die Fähigkeit, die Bilddaten von der Kamera zu laden und diese durch die Algorithmen zu verarbeiten, bevor die nächsten Daten geladen werden und dabei eine Bildrate von mindesten 24 Bildern pro Sekunde zu realisieren. Wie in Kapitel 5.1.1 beschrieben, und unter der hypothetischen Annahmen, dass die Bilddaten unendlich schnell geladen werden, bedeutet dies, dass den Algorithmen maximal 0,042 Sekunden Bearbeitungszeit zur Verfügung stehen.

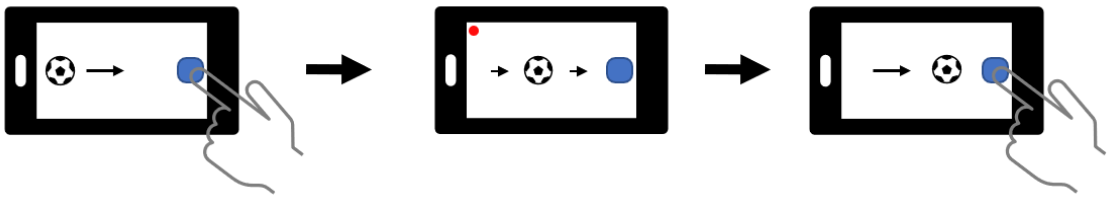

Abbildung 6.8: Ausführen der Messung

Nach erfolgter Messung erfolgt die Ausgabe des Ergebnisses und die Möglichkeit dieses zur Spielerverwaltung hinzuzufügen. Dies wird durch ein Pop-up-Fenster geschehen.

# 7 Implementierung der Bildverarbeitung

In diesem Kapitel wird die Entwicklung der Bildverarbeitung näher betrachtet. Zunächst wird auf die Vorbereitung der Entwicklung in Form der Aufnahme der zu analysierenden Videos eingegangen. Des Weiteren ist dieses Kapitel, entsprechend der zwei zu entwickelnden Algorithmen, in zwei weitere Unterkapitel aufgeteilt. Zunächst einmal wird es um die Entwicklung des allgemeinen Algorithmus gehen. Im zweiten Teil dieses Kapitels wird dann der optimierte Algorithmus betrachtet.

Wie im Konzept erwähnt, findet die Implementierung in Python statt. Genauer gesagt wird die Python Version 3.7.9 in der Spyder IDE[1] verwendet. Beide Algorithmen werden modular in Funktionen programmiert, sodass diese Funktionen zu einem späteren Zeitpunkt einzeln übersetzt und in die App eingefügt werden können. Bei der Programmierung wird der Aufruf als native Funktion, wie er in der App erfolgen würde, simuliert. Das bedeutet, dass eine Grundstruktur im Python Skript entwickelt wird, die so auch in Java implementiert werden kann und aus der aus die entsprechenden Funktionen aufgerufen werden. Die eigentliche Bildverarbeitung findet dann ausschließlich in ausgelagerten Funktionen statt, die aus der Grundstruktur heraus aufgerufen werden (siehe Abbildung 7.1). Es dürfen somit keine globalen Variablen in den modular programmierten Funktionen verwendet werden und diese können nur über Rückgabewerte und Aufrufparameter mit der Struktur kommunizieren. Die Funktionen dürfen zudem nur einen Rückgabeparameter haben, mit der Ausnahme, dass es sich um primitive Datentypen handelt, die in C++ als zusammengesetzter String übergeben werden können.

[1] Siehe: https://www.spyder-ide.org/ [Letzter Zugriff: 19.02.2021]

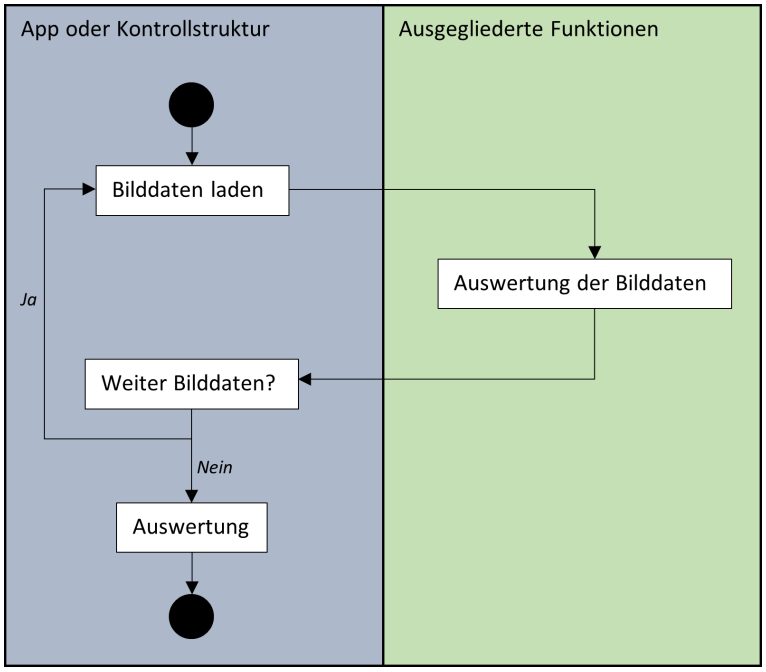


Abbildung 7.1: Simulierte Trennung von Funktionen und Hauptablauf

## 7.1 Videos aufnehmen und präparieren

Um schon vor der Fertigstellung einer App einen Algorithmus entwickeln und testen zu können, werden Videos aufgenommen, die am Computer ausgewertet werden. Um möglichst viele verschiedene Situationen betrachten zu können, sind die Videos an unterschiedlichen Orten und unter unterschiedlichen Bedingungen entstanden. Gefilmt wurde an drei verschiedenen Fußballplätzen. Einige der Videos wurden mit einer Kamera und einem Stativ aufgenommen, andere wurden mit dem Handy und aus der Hand aus aufgenommen. Um auch testen zu können, wie mit Fehlern umgegangen wird, wurde zusätzlich auf einigen Videos daneben geschossen. Variiert wurden auch die Schussstärke und Höhe, um möglichst viele denkbare Situationen abbilden zu können. Um herausfinden zu können, wie sehr die Bilder verwackelt sein können, bevor der Schuss nicht mehr nachverfolgt werden kann, wurden einige Videos auch absichtlich verwackelt.

Zum Testen des optimierten Algorithmus müssen einige Daten aus den Videos bekannt sein, etwa die Ballposition im ersten Bild oder die Position der Torpfosten (vgl. Kapitel 6.1). Diese werden aus dem jeweils ersten Bild der aufgenommenen Videos ausgelesen.

Das erste Bild wird mit einem Python Skript aus den Videos als separates Bild abgespeichert und per Hand ausgewertet. Die benötigten Daten werden dann in Textdateien gespeichert, die dann vom Python Skript, in dem die Bildverarbeitung stattfindet, geladen werden können. So werden die Eingaben, die der Nutzer vorzunehmen hat, simuliert. Die Tabelle 7.1 zeigt die Daten, die in der Textdatei gespeichert sind. Diese stammen beispielsweise aus der Videodatei 0101.*mp*4. Bekannt sein müssen die Koordinaten des Bildausschnitt, der den Ball enthält, sowie die der Pfosten, wie im Konzept der Bildverarbeitung (Kapitel 6.1) beschrieben wurde. Zusätzlich muss die Schussdistanz und die Bildrate bekannt sein. Die Schussdistanz muss in der App vom Nutzer eingegeben werden. Für die aufgezeichneten Videos wurde die Entfernung beim Aufzeichnen der Videos gemessen. Die Bildrate muss in der App, beim Speichern der Bilddaten, bestimmt werden. Für die Videos ist sie aus den Bilddaten bekannt.

| Wert | Bezeichnung |
|---|---|
| 108 | Start der Ball-Box |
| 601 | Start der Ball-Box |
| 154 | Ende der Ball-Box |
| 637 | Ende der Ball-Box |
| 1568 | erster Torpfosten |
| 1748 | zweiter Torpfosten |
| 9.9 | Entfernung zum Tor |
| 30 | Bildrate des Videos |

Tabelle 7.1: Daten der Textdatei beim optimierten Algorithmus

Für die Berechnung der Geschwindigkeit mit dem allgemeinen Algorithmus müssen nur die mittlere Entfernung der Kamera zur Flugbahn des Balles, der Kamerawinkel und die Bildrate bekannt sein. Auch diese Daten wurden in einer Textdatei gespeichert, sodass die Nutzereingabe simuliert werden kann. Tabelle 7.2 zeigt dafür ein Beispiel anhand der Videodatei 0201.*mp*4.

| Wert | Bezeichnung |
|---|---|
| 10 | Entfernung der Kamera zur Fluglinie |
| 70.0 | Kamerawinkel |
| 30 | Bildrate des Videos |

Tabelle 7.2: Daten der Textdatei beim allgemeinen Algorithmus

Sämtliche Videos wurden, wie im Konzept festgelegt, in Full-HD aufgenommen.

## 7.2 Implementierung des allgemeinen Algorithmus

Wie beschrieben, wird die Implementierung durch ausgelagerte Funktionen vorgenommen. Für den allgemeinen Algorithmus ist nur eine Funktion nötig. Dieser werden zwei aufeinanderfolgende Bilder übergeben. Zurückgegeben werden die Koordinaten des detektierten Ballmittelpunktes, wenn einer detektiert wird. Diese Koordinaten werden in einer Liste gespeichert. Nachdem das letzte Bild ausgewertet ist, findet die Auswertung der Ballpositionen statt. Die Auswertung muss nicht zwangsläufig in eine native Funktion ausgelagert werden, da diese auch in Java implementierbar wäre.

Wie im Konzept in Kapitel 6.1 beschrieben, wird die Detektion des Balles nicht im gesamten Bild, sondern nur in einem mittigen Ausschnitt stattfinden. Dieser wurde nach eigenem Ermessen festgelegt. Bei der Überlegung nach der Größe ist einerseits zu bedenken, dass dieser Ausschnitt möglichst groß ist, um viel Fläche zur Detektion zu bieten (bei einem großen Bereich ist es einfacher, diesen mit dem Ball zu treffen). Andererseits ist an den Kanten noch ein Pufferbereich vorzusehen, der wichtig ist, damit das Bild besser ausgerichtet werden kann, ohne, dass Bewegungen am Rand unbeabsichtigt detektiert werden. Es ist festgelegt, dass dieser Pufferbereich 430 Pixel vertikal (je rechts und links) und 200 Pixel horizontal (unten und oben) beträgt. Der Detektionsbereich erstreckt sich somit über die Koordinaten 430 bis 1490 vertikal ($x$) und 200 bis 880 horizontal ($y$). Abbildung 7.2 zeigt ein Beispiel.


Abbildung 7.2: Bildausschnitt allgemeiner Algorithmus

### 7.2.1 Detektion

Es wird das Differenzbild des Suchbereiches gebildet. Die Detektion des Balles in diesem Suchbereich basiert darauf, dass die Zeile und die Spalte des Bildausschnittes gefunden wird, in der (im Vergleich zum letzten Bild) die meisten Bewegungen stattgefunden haben. Dazu werden die Summen der Pixel-Werte für alle Zeilen und Spalten in einer Liste gespeichert, wie im Quelltexte-Auszug 7.1 zu sehen ist (Zeilen 1-4 und 9-11).

```
1  pixSumOfRow = []
2
3  for i in range(10, max_y - min_y - 10):
4      pixSumOfRow.append( np.sum(frame[i,:]) )
5
6  maxposition_y = pixSumOfRow.index( max(pixSumOfRow) )
7
8  pixSumOfColumn = []
9
10 for i in range(10, max_x - min_x - 10):
11     pixSumOfColumn.append( np.sum(frame[:,i]) )
12
13 maxposition_x = pixSumOfColumn.index( max(pixSumOfColumn) )
```

Listing 7.1: Detektion des Bewegungsmaximums

In den Zeilen 6 und 13 werden dann die maximalen Werte bestimmt. Diese Koordinaten werden als detektierter Mittelpunkt behandelt. Durch Verwacklungen, besonders bei horizontalen oder vertikalen Strukturen, kann es dazu kommen, dass falsche Detektionen entstehen, wie in Abbildung 7.4 zu sehen ist. Um dem entgegenzuwirken, wird in einem ersten Schritt möglichst viel von diesem Hintergrundrauschen weg gefiltert. Dazu wird genutzt, dass es sich bei Aktivierungen durch Verwacklungen vor allem um Umrisse handelt. In Abbildung 7.3 ist auf der linken Seite der Auszug aus einem Bild zu sehen, in dem Aktivierungen durch ein Verwackeln der Kamera entstanden sind. Auf der rechten Seite ist die Aktivierung durch einen Ball zu sehen. Der Unterschied ist, dass bei der Aktivierung durch einen Ball vertikal und horizontal in etwa dieselbe Pixelanzahl aktiviert ist. Bei den Aktivierungen, die durch die Verwacklung resultieren, handelt es sich um Linien, die vor allem in eine Richtung stark ausschlagen.

Abbildung 7.3: Unterschied Ball und störende Umrisse

Durch Anwendung eines Filters wird das Bild künstlich unscharf gemacht. Danach werden Aktivierungen, die einen gewissen Schwellenwert unterschreiten auf 0 gesetzt. Dies wird zweimal hintereinander angewandt, wie im Quelltext-Auszug 7.2 zu sehen ist.

```
1  frame = cv2.absdiff(frame1 , frame2)
2
3  frame = cv2.filter2D(frame, -1, KERNEL_BLUR)
4  frame = cv2.filter2D(frame, -1, KERNEL_BLUR)
5
6  ret, frame = cv2.threshold(frame, 50, 255, cv2.THRESH_BINARY)
7
8  frame = cv2.filter2D(frame, -1, KERNEL_BLUR)
9  frame = cv2.filter2D(frame, -1, KERNEL_BLUR)
10
11 ret, frame = cv2.threshold(frame, 200, 255, cv2.THRESH_BINARY)
```

Listing 7.2: Filtern der Hintergrundbewegungen

Die verwendeten Werte wurden experimentell, anhand der zum Entwickeln aufgenommenen Videos, bestimmt. Das Resultat ist in der Abbildung 7.5 zu sehen. Zusätzlich dazu wird die letzte bekannte Detektion mit einem schwarzen Kasten übermalt, damit die Doppelerscheinungen der Bälle nicht zu falschen Detektionen führen. Die Abbildungen 7.4 und 7.5 zeigen das gleiche Differenzbild; 7.4 ohne und 7.5 mit den erwähnten Korrekturen. Zu erkennen ist, wie der Ball erst nach den Korrekturen richtig erkannt wird.

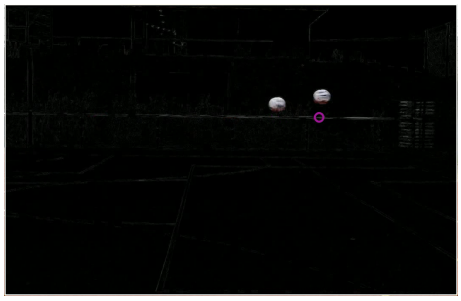

Abbildung 7.4: Falsche Detektion beim allgemeinen Algorithmus

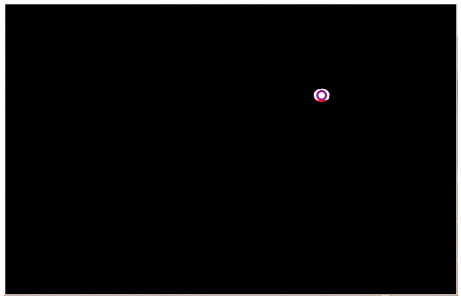

Abbildung 7.5: Richtige Detektion beim allgemeinen Algorithmus

Zudem wird die detektierte Koordinate zwei weiteren Überprüfungen unterzogen, wie im Konzept (Kapitel 6.1) beschrieben wurde. Zunächst erfolgt eine Überprüfung, ob ein gewisser Bereich um den detektierten Punkt einen festgelegten Schwellwert überschreitet, wie im Quelltext-Auszug 7.3 zu sehen ist. Auch diese Maßnahme verringert die Detektion von Umrissen.

```
1  meanInArea = np.mean( frame[ maxposition_y-10:maxposition_y+10 ,
2                          maxposition_x-10:maxposition_x+10 ] )
3  if( math.isnan(meanInArea) or meanInArea <= 40 ):
4        ...
```

Listing 7.3: Umliegende Aktivierung überprüfen

Auch hier wurden die Werte durch eine experimentelle Überprüfung gewählt. Die Wahl der Größe des Ausschnittes (Zeile 1) hat eine direkte Auswirkung auf die mögliche Detektion von kleineren Bällen. Ein groß gewählter Bereich filtert mehr Umrisse, bedeutet aber

auch, dass kleinere Bälle nicht mehr erkannt werden[2]. Ein kleiner Bereich lässt auch die Detektion von kleineren Bällen zu, erhöht aber die Gefahr, dass Umrisse detektiert werden. Die hier verwendeten Parameter funktionieren beispielsweise hervorragend für die Detektion von Fußbällen, doch detektieren Tennisbälle nicht[3]. Die Wahl dieser Parameter muss je nach Anwendung variieren.

Zusätzlich findet eine Überprüfung statt, ob es sich bei der detektierten Koordinate auch um den Punkt eines Kreises handelt. Dazu wird genutzt, dass bei der Aktivierung durch einen Ball vertikal und horizontal in etwa die gleiche Pixelanzahl aktiviert ist (vgl. Abbildung 7.3). Um herauszufinden, ob es sich um einen Kreis handelt, wird von der detektierten Koordinate aus die horizontale und die vertikale Größe der Aktivierung gemessen. Dazu werden von den Koordinaten aus die aktivierten Pixel nach unten, oben, rechts und links gezählt (vgl. Quelltext-Auszug 7.4 als Beispiel für die Pixel nach rechts).

```
1  try:
2        i = maxposition_x + 10
3        while(np.sum(img[maxposition_y + 10][i]) > 10.0):
4              i += 1
5  except:
6        i -= 1
7
8  x_right = i - 1 - maxposition_x - 10
```

Listing 7.4: Größe der Aktivierung messen

Dabei hat sich ein Schwellwert von 10 für die Aktivierung als effektiv erwiesen. Das Ergebnis ist in Abbildung 7.3 auf der rechten Seite zu sehen. Wenn der Faktor zwischen der vertikalen und der horizontalen Aktivierung kleiner ist als 1.5, so wird angenommen, es handelt sich bei der Aktivierung um einen Kreis, also um einen Ball.

```
1  xg = x_left + x_right
2  yg = y_up + y_down
3
4  if(xg / yg < 1.5 and yg / xg > 0.67):
5        ...
```

Listing 7.5: Auf Kreisform überprüfen

[2]Das Problem dabei ist, dass sich kleine Bälle weniger deutlich von einem verwackelten Hintergrund abheben. Dadurch wirkt sich ein starkes Filtern immer direkt auf die Detektion der Bälle aus.
[3]Bei einer mittleren Entfernung zur Flugbahn von 8 Metern.

In diesem Fall wird die detektierte Koordinate beibehalten; andernfalls gibt die Funktion die Information zurück, dass die Detektion nicht erfolgreich war.
Durch Bewegungen im Hintergrund oder Verwacklungen der Kamera kann es zu weiteren Aktivierungen kommen, die dafür sorgen, dass der Kreismittelpunkt nicht richtig getroffen wird. Auch durch Schatten im Bild oder durch das Wegfiltern zu Anfang der Funktion kann es dazu kommen, dass der Ball nicht exakt kreisförmig abgebildet wird und so der Ball nicht mittig detektiert wird. Für den Fall, dass ein Kreis detektiert wurde, wird die Koordinate zusätzlich weiter in den Kreismittelpunkt gesetzt, wie in Abbildung 7.6 zu sehen ist. Dieser Vorgang wird nachfolgend als Formkorrektur bezeichnet.

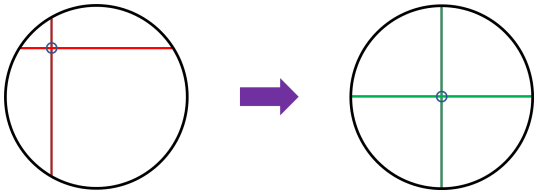

Abbildung 7.6: Detektierte Koordinate in den Kreismittelpunkt versetzten

Es werden die Bezeichnungen, wie in Abbildung 7.7 rechts zu sehen, eingeführt. Zudem werden die Koordinaten der detektierten Koordinate als $x_{ist}$ und $y_{ist}$ und die Koordinaten des gesuchten Mittelpunktes als $x_{mitte}$ und $y_{mitte}$ bezeichnet.

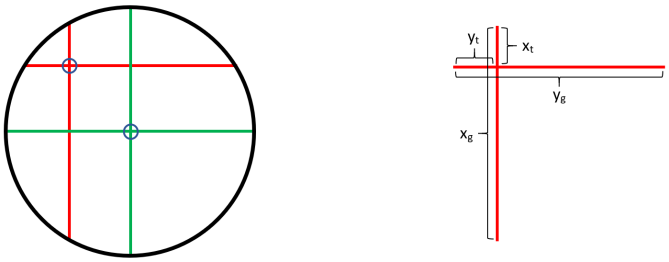


Abbildung 7.7: Grundlage zur Berechnung der Formkorrektur

Es ist festzustellen, dass die grünen Mittelpunkts-Linien die gemessenen (roten) Linien immer genau in der Mitte schneiden, also bei:

$$\frac{x_g}{2} \text{ und bei } \frac{y_g}{2} \tag{7.1}$$

Daraus lässt sich der neue Mittelpunkt berechnen:

$$x_{mitte} = x_{ist} + \left(\frac{xg}{2} - x_t\right) \tag{7.2}$$

$$y_{mitte} = y_{ist} + \left(\frac{y_g}{2} - y_t\right) \tag{7.3}$$

Abbildung 7.8 zeigt ein Beispiel einer Detektion mit und ohne Formkorrektur. Zu sehen ist sehr gut, wie der Mittelpunkt besser getroffen wird.

Abbildung 7.8: Testergebnis der Formkorrektur

### 7.2.2 Auswertung

Die Auswertung erfolgt, wie beschrieben, auf Grundlagen der mittleren Entfernung zur Flugbahn des Balles und des Kamerawinkels. Es wird für je zwei aufeinanderfolgende Punkte die dazwischen befindliche Strecke und daraus (in Verbindung mit der Bildrate) die Geschwindigkeit berechnet. Die folgende Abbildung zeigt beispielhaft für zwei links der Mitte liegende Punkte, wie die Strecke zwischen diesen Punkten berechnet wird.

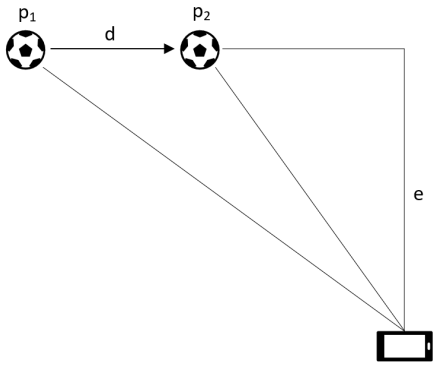


Abbildung 7.9: Analyse der Datenpunkte

Wie im Konzept beschrieben, werden Full-HD Videos verwendet, dementsprechend befindet sich die Mitte des Bildes bei 960 Pixel. Die folgenden Berechnungen zeigt, wie die Distanz $d$ zwischen zwei Punkten auf der Flugbahn durch die detektierten Punkte im Bild $p_1$ und $p_2$ berechnet werden kann. Dabei sind die Distanz $d$ und die mittlere Entfernung zur Flugbahn $e$ in Metern und $p_1$ und $p_2$ in Pixeln angegeben. Zudem sei $\alpha$ der Bildwinkel der Kamera. So lässt sich $d$, bei zwei links der Mitte liegenden Punkten wie folgt berechnen:

$$d = e \cdot \left( tan\left( \frac{960 - p_1}{1920} \cdot \alpha \right) - tan\left( \frac{960 - p_2}{1920} \cdot \alpha \right) \right) \tag{7.4}$$

Und bei zwei Punkten auf der rechten Seite:

$$d = e \cdot \left( tan\left( \frac{p_2 - 960}{1920} \cdot \alpha \right) - tan\left( \frac{p_1 - 960}{1920} \cdot \alpha \right) \right) \tag{7.5}$$

Bei zwei Punkten, die auf unterschiedlichen Seiten der Mitte liegen, wird unter der Näherung, dass beide Punkte symmetrisch um die Mitte liegen würden, folgende Rechnung durchgeführt:

$$d = 2 \cdot e \cdot tan\left( \frac{p_2 - p_1}{2 \cdot 1920} \cdot \alpha \right) \tag{7.6}$$

Analog dazu wird die vertikale Distanz berechnet. Mit der Information der Zeitspanne zwischen zwei Bildern kann die Geschwindigkeit berechnet werden. Der Quelltext-Auszug

7.6 zeigt dies anhand zweier Koordinaten links der Mitte.

```
1  dist = DISTANCE * ( math.tan((960 - lp[0]) * AnglePerPix) -
2             math.tan((960 - p[0]) * AnglePerPix) )
3  speedx = dist / (SecondsPerPics * (p[2] - lp[2]) )
```

Listing 7.6: Horizontale Geschwindigkeit berechnen

Beide Geschwindigkeitsteile (vertikale und horizontale Bewegung) werden über den Satz des Pythagoras zusammengefügt.

```
1  speedxy =  math.sqrt( speedx**2 + speedy **2 )
```

Listing 7.7: Gesamte Geschwindigkeit berechnen

Aus allen einzelnen Geschwindigkeiten (zwischen zwei aufeinanderfolgenden Punkten) wird der Mittelwert gebildet.


Abbildung 7.10: Ausgabe der Analyse

### 7.2.3 Test

Um die Geschwindigkeitsmessung zu überprüfen, werden erneut Videos aufgezeichnet, wobei die Geschwindigkeit gleichzeitig mit dem Radar-Messgerät aus 3.1 gemessen wird, um die Geschwindigkeiten vergleichen zu können. Die Tabelle 7.3 enthält fünf repräsentative Ergebnisse.

Zu erkennen ist, dass die mit dem Algorithmus gemessenen Werte allesamt größer sind als die, des Messgerätes. Zwar liegen die Werte in der gleichen Größenordnung, doch sind die Unterschiede, mit einem Faktor von bis zu 1.3, beträchtlich. Eine Erklärung dafür

| Video | Radar-Messgerät | allgemeiner Algorithmus |
|---|---|---|
| 0202 | 55 | 67,8 |
| 0203 | 54 | 59,3 |
| 0204 | 44 | 58,2 |
| 0205 | 47 | 52,8 |
| 0210 | 52 | 64,0 |

Tabelle 7.3: Testergebnisse allgemeiner Algorithmus (Messungen in km/h)

wäre die, in Kapitel 3.1 bereits angesprochene, Tatsache, dass das Radar-Messgerät die zusätzliche vertikale Bewegung nicht messen kann. Die Messungen wären somit immer kleiner gleich denen des Algorithmus. Betrachtet man die Unterschiede genauer, so lässt sich feststellen, dass die Unterschiede größer sind, umso größer die Aufwärtskomponente des Schusses ist, wie in der Tabelle 7.4 festgehalten. Dabei gilt:

$$\text{Faktor} = \frac{\text{Messwert Algorithmus}}{\text{Messwert Radar}} \tag{7.7}$$

| Video | Schusswinkel $\alpha$ in $\circ$ | Faktor |
|---|---|---|
| 0202 | 20 | 1,233 |
| 0203 | 6 | 1,098 |
| 0204 | 22 | 1,327 |
| 0205 | 15 | 1,123 |
| 0210 | 25 | 1,23 |

Tabelle 7.4: Messunterschiede und Schusswinkel

Diese Ergebnisse legen nahe, dass die angesprochene Unfähigkeit des Radar-Messgeräts die vertikale Bewegung zu messen, tatsächlich der Grund für die Unterschiede ist. Um den Algorithmus nochmals zu testen, wurden die Koordinaten der Ballpositionen per Hand ausgewertet, um daraus mit derselben Berechnung, die hinter dem Algorithmus steckt, die Geschwindigkeit zu berechnen. Die Ergebnisse gleichen, bis auf marginale Unterschiede, denen des Algorithmus, womit der Anforderung N8 genüge geleistet ist. Die Ergebnisse zeigen zudem, dass die Genauigkeit der Messergebnisse die des Radar-Messgerätes bei weitem überstiegen. Zur Veranschaulichung zeigt die Abbildungen 7.11 Bilder aus zwei der verwendeten Videos. Oben zu sehen ist Video 0204 und unten Video 0210.


Abbildung 7.11: Bilder aus den Testvideos

## 7.3 Implementierung des optimierten Algorithmus

Da die Arbeit vorrangig die Entwicklung eines Algorithmus betrachtet, der die Geschwindigkeiten von Fußbällen misst, gilt diesem Kapitel ein besonderes Augenmerk. Wie auch der allgemeine Algorithmus wird auch dieser in auslagerbaren Funktionen entwickelt. Dabei gelten die Regeln, wie sie zu Anfang in Kapitel 7 beschrieben sind.

### 7.3.1 Schusserkennung

Wie im Konzept (Kapitel 6.1) beschrieben, muss der Nutzer den Bereich, in dem sich der Ball befindet, manuell markieren. Dieser Bereich wird in jedem Bild überprüft, ob ein Schuss stattgefunden hat. Dazu wird genutzt, dass sich der Ball nicht bewegt, bis er geschossen wird. Dann kommt es zu einer plötzlichen Bewegung. Zunächst wird aus dem Bereich zweier aufeinanderfolgender Bilder, in denen sich der Ball befindet, das Differenzbild gebildet. Dann werden die aktivierten Pixel im Bildausschnitt gezählt. Dazu wird in einer Schleife über alle Pixel ausgewertet, ob diese einen experimentell gewählten

Schwellwert von 20 überschreiten. Falls dies zutrifft, werden sie als aktiviert angesehen. Übersteigt die Zahl der aktivierten Pixel die der inaktivierten um den Faktor 1.2, so wird angenommen, dass der Ball in diesem Bild geschossen wurde.

```
1  for row in img:
2      for pixel in row:
3          if((int(pixel[0]) + int(pixel[1]) + int(pixel[2])) > 20):
4                  pixHight += 1
5          else:
6                  pixLow += 1
7
8  if (pixHight >= 1.2 * pixLow):
9      return True
10 else:
11     return False
```

Listing 7.8: Detektion des Schussbeginns

### 7.3.2 Nachverfolgung der Ballpositionen

Nach Beginn des Schusses werden die Ballpositionen verfolgt. Damit möglichst wenig Bewegungen in der Umgebung die Detektion negativ beeinflussen, wird der Suchbereich, in dem sich der Ball befinden muss, eingegrenzt. Dabei unterscheidet sich die Eingrenzung des Suchbereiches je nachdem, wie weit der Schuss fortgeschritten ist. Zu Anfang des Schusses, wenn der Schussverlauf noch nicht genau eingeschätzt werden kann, wird der Suchbereich unabhängig von der Geschwindigkeit des Balles eingegrenzt. Erst nach einigen detektierten Positionen wird der Suchbereich durch eine Trendanalyse der vorherigen Positionen erweitert. Durch Ausprobieren verschiedener Einstellungen, hat sich herausgestellt, dass ein Hinzuziehen der Trendanalyse nach drei detektierten Ballpositionen am effektivsten ist. Die Berechnung des Suchbereiches ohne Trendanalyse ist in Quelltext-Ausschnitt 7.9 zu sehen. Der Bildausschnitt wird durch die Koordinaten ($min\_x$,$min\_y$) und ($max\_x$,$max\_y$) eingegrenzt. Eine weitere Unterscheidung erfolgt durch die Richtung in die geschossen wird. Die Variablen „BALL_BOX_SIZE_X“ und „BALL_BOX_SIZE_Y“ beschreiben die Größe des Bildausschnittes in dem, sich der Ball vor Beginn des Schusses befunden hat.

```
1  max_y = int(lastY + BALL_BOX_SIZE_Y / 3)
2  min_y = lastY - 2 * BALL_BOX_SIZE_Y
3  ...
4
5  if(LEFT_TO_RIGHT):
```

```
6             min_x = int(lastX + BALL_BOX_SIZE_X / 2)
7             max_x = int(lastX + 4 * BALL_BOX_SIZE_X)
8         ...
9
10    else:
11            max_x = int(lastX - BALL_BOX_SIZE_X / 2)
12            min_x = int(lastX - 4 * BALL_BOX_SIZE_X)
13        ...
```

Listing 7.9: Eingrenzung des Suchbereichs

Damit die Koordinaten nicht die Bilddimensionen überschreiten, findet diesbezüglich eine Überprüfung statt. Diese wurde im vorliegenden Quelltext-Auszug weggelassen und durch „...“ angedeutet.
Die Trendanalyse besteht darin, dass zunächst eine theoretische Position berechnet wird, an der sich der Ball in diesem Bild befinden müsste. Diese Position wird anhand der letzten bekannten Position und der Differenz zu der Vorherigen berechnet. Um diese Position wird der Suchbereich gebildet, wie im Auszug 7.10 zu sehen ist.

```
1   trendY = lastPosition[1] - lastLastPosition[1]
2   trendX = lastPosition[0] - lastLastPosition[0]
3
4   nextPosX = lastPosition[0] + trendX
5   nextPosY = lastPosition[1] + trendY
6
7   max_y = newxtPosY + BALL_BOX_SIZE_Y
8   min_y = newxtPosY - BALL_BOX_SIZE_Y
9   ...
10
11  if(LEFT_TO_RIGHT):
12        min_x  = int( newxtPosX - 0.75 * BALL_BOX_SIZE_X )
13        max_x  = int( newxtPosX + 1.5 * BALL_BOX_SIZE_X )
14        ...
15  else:
16        min_x  = int( newxtPosX - 1.5 * BALL_BOX_SIZE_X )
17        max_x  = int( newxtPosX + 0.75 * BALL_BOX_SIZE_X )
18        ...
```

Listing 7.10: Eingrenzung des Suchbereichs durch Trendanalyse

Die Abbildung 7.12 zeigt den eingegrenzten Suchbereich (blauer Kasten), oben mit und unten ohne Trendanalyse. Die Größe des Suchbereiches zu Anfang wurde so gewählt, damit (unter Berücksichtigung von Kapitel 5.1.1) die Anforderung N5 erfüllt wird.

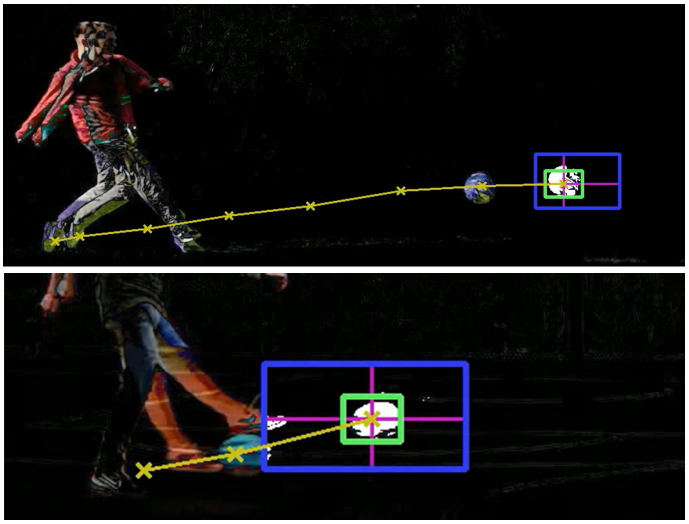

Abbildung 7.12: Eingrenzung des Suchbereiches[4]

### 7.3.3 Detektion des Balles

Um den Ball in diesem Suchbereich zu finden, wird das Differenzbild dieses Bereichs zwischen dem aktuellen und dem letzten Bild gebildet. Die Detektion läuft dabei ähnlich wie die im allgemeinen Algorithmus. Nach Bildung des Differenzbildes wird die Filterung wie in Auszug 7.2 durchgeführt. Anschließend wird eine Aktivierungsfunktion angewandt, die alle Pixel, deren Summe den Wert 120 übersteigt, auf einen Wert von 255 setzt. Dies hat sich bei Tests als sinnvoll erwiesen, da so schwerwiegendere Differenzen zusätzlich gewichtet werden.
Anschließend werden Summen der Zeilen und Spalten, vergleichbar wie beim allgemeinen Algorithmus in Auszug 7.1, in einer Liste gespeichert. Dabei findet eine Gewichtung der umliegenden Zeilen und Spalten statt. Dies hat den Grund, dass so zusätzlich der Einfluss der Umrisse im Hintergrund reduziert wird. Dann wird, wie beim allgemeinen Algorithmus, das Maximum aus Zeilen und Spalten gesucht, welches als Koordinate des Balles angesehen wird.

[4]Die Bilder stammen aus älteren Versionen der Bildverarbeitung, in denen das Differenzbild noch vom ganzen Bild berechnet wurde. Um Laufzeit zu sparen wird in der letztendlichen Version dies nur von dem Suchbereich berechnet. Die dargestellte Version eignete sich jedoch am besten zur Darstellung des eingegrenzten Suchbereiches.

```
1  for i in range(0, len(sumPerColumn) - 4):
2        diffDataArray.append(sumPerColumn[i + 2] +
3                             0.5 * (sumPerColumn[i + 1] + sumPerColumn[i + 3]) +
4                             0.25 * (sumPerColumn[i] + sumPerColumn[i + 4]))
5
6  maxposition_y = diffDataArray.index(max(sumPerColumn)) + 2
```

Listing 7.11: Gewichtetes Summieren der Zeilen

Bei der Suche nach dem Maximum in den Spalten ergibt sich eine Besonderheit. Um nicht fälschlicherweise Teile vom Bein oder die vorangegangene Doppelerscheinung des Balles zu detektieren, wird bei der horizontalen Suche nach dem Maximum nicht der gesamte Bildbereich durchsucht. Es wird nur in den hintersten 60 Prozent der Aktivierungen gesucht. Es werden dazu die gesamten Pixeldaten summiert und nur die Spalten betrachtet, bei denen die vorherigen Spalten zusammen 40 Prozent der aktivierten Pixel beinhalten. Um bei dieser Berechnung nicht zu weit nach hinten zu rutschen, wird mindestens die hintere Hälfte der Pixeldaten betrachtet. Das Prinzip wird durch Abbildung 7.13 verdeutlicht. Die blauen Kästen stellen die aktivierten Pixel dar, die roten Pfeile zeigen an die Stelle, ab der 40 Prozent der Aktivierungen überschritten sind und die grünen Pfeile zeigen auf die Position, ab der die Suche nach dem maximalen Wert beginnt.

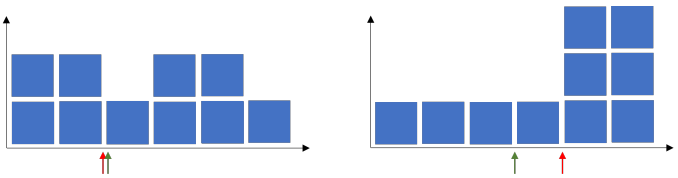

Abbildung 7.13: Schema der Suche nach dem Maximum in den Spaltenwerten

Es unterscheiden sich dabei die Berechnungen je nach Schussrichtung. Der Quelltext-Auszug 7.12 zeigt die Berechnung bei einem Schuss von links nach rechts.

```
1  completeArray = np.sum(sumPerRow)
2  lenData = len(sumPerRow)
3
4  for s in range(0, lenData):
5          if (np.sum(sumPerRow[0:s]) > completeArray * 0.4):
6            break
7
8  if(s > 0.5 * lenData):
9        s = int(0.5 * lenData)
10
```

```
11  for i in range(s, lenNew):
12      diffDataArray.append(sumPerRow[i + 2] +
13                  0.5 * (dsumPerRow[i + 1] + sumPerRow[i + 3]) +
14                  0.25 * (sumPerRow[i] + sumPerRow[i + 4]))
```

Listing 7.12: Gewichtetes Summieren der Spalten

Dieses Verfahren hat sich bei der Entwicklung des Algorithmus als besonders effektiv erwiesen. Gerade bei den ersten Detektionen, bei denen es sein kann, dass noch Teile vom Bein mit im Bild sind, hat es die Fehlerquote deutlich reduzieren können. Abbildung 7.14 zeigt ein Bild aus dem Video 0101.*mp*4, bei dem gerade die Ballpositionen nachverfolgt werden.

Abbildung 7.14: Nachverfolgung der Ballpositionen beim optimierten Algorithmus

Während der Verfolgung des Schusses kommt es zu mehreren Plausibilitätskontrollen der detektierten Koordinaten. Zum einen kann es dazu kommen, dass die aktuelle Position perspektivisch[5] hinter der letzten liegt. Da es nicht passieren kann, dass der Ball mitten im Flug die Richtung wechselt, wird dies als Fehler aufgefasst. Zu diesem Fehler kam es während der Entwicklung immer wieder dann, wenn beim Abschuss nicht der Ball, sondern ein Teil vom Bein erkannt wird. Zum anderen kann es dazu kommen, dass die Änderungen der $y$-Positionen größer ist, als die Änderungen der $x$-Positionen, was nach Kapitel 5.1.2 als ausgeschlossen angesehen wird. Aus diesem Grund gilt auch dies als Indikation für einen Fehler. Kommt es zu einem dieser Fehler, wird die Bildverarbeitung abgebrochen und die Information ausgegeben, dass ein Fehler aufgetreten ist. Das Schussende wird erkannt, sobald die x-Koordinate der detektierten Ballposition hinter der x-Position des näheren Torpfostens ist. Im Anschluss daran kommt es zur Auswertung der Ballpositionen. Für den Fall, dass die Verfolgung nach oben oder unten außerhalb des Bildbereiches verläuft, wird ebenfalls von einem Fehler ausgegangen.

[5] Abhängig davon, in welche Richtung der Schuss stattfindet.

### 7.3.4 Auswertung

Die Auswertung der Ballpositionen erfolgt aus der Schussdistanz in Metern und der Differenz in Pixeln zwischen Ball und Tor. Letztere wird aus den Markierungen entnommen, die der Nutzer, wie in Kapitel 6.1 beschrieben, vorzunehmen hat. Die Geschwindigkeit wird mit einem Dreisatz berechnet. Aus diesen Werten wird (unter der vereinfachenden Annahme, die Entfernung sei über die Pixel gleich verteilt) die Distanz pro Pixel berechnet[6]. Die Berechnung der zurückgelegten Distanz durch den beschriebenen Dreisatz ist nicht gänzlich korrekt, da die dafür angenommene Linearität in der Realität nicht gegeben ist[7]. In Anbetracht der in Kapitel 5 berechneten Messungenauigkeiten werden diese Berechnungsungenauigkeiten jedoch vernachlässigen, da es hier für den Nutzer einfach gemacht werden soll, sodass dieser nur die Distanz eingeben muss. Aus der horizontalen und vertikalen Differenz zwischen zwei Punkten wird mit dem Satz des Pythagoras die Distanz der direkten Verbindung der beiden Punkte berechnet. Aus diesem Wert und der Bildrate wird die Geschwindigkeit für jedes Paar aufeinanderfolgender Positionen berechnet. Der endgültige Wert der Analyse bildet der Mittelwert all dieser Werte. Der verkürzte Quelltext-Auszug 7.13 zeigt diese Berechnung.

```
1  numberDatapoints = len(ballPositions)
2  ...
3  for P in ballPositions:
4      ...
5      deltaX = P[0] - lastP[0]
6      deltaY = P[1] - lastP[1]
7      ...
8      deltaD = math.sqrt(deltaX * deltaX + deltaY * deltaY) *
9          (DISTANZE_SCHUSS / PIX_DIST)
10     meanSpeed += (deltaD / TIME_BETWEEN_FRAMES) / (numberDatapoints - 1)
11     ...
```

Listing 7.13: Auswertung der Ballpositionen

Abbildung 7.15 zeigt die Videos 0101.*mp*4 (oben) und 0022.*mp*4 (unten) nach Ende des Schusses und erfolgter Auswertung.

[6] Gemeint ist die Annahmen, dass ein Pixel immer die gleiche Distanz repräsentiert, egal wo im Bild dieser auftaucht.

[7] Durch die Optik der Kameralinsen repräsentieren Pixel eine umso größere Distanz je weiter außen sie liegen.


Abbildung 7.15: Auswertung der Ballpositionen beim optimierten Algorithmus

### 7.3.5 Tracking-Algorithmus

Bei Tests mit aus der Hand aufgenommenen Videos hat sich gezeigt, dass die dabei entstehenden minimalen Verwacklungen schon ausreichen, um die Schusserkennung auszulösen. Aus diesem Grund wurde bei fast allen dieser Videos Schüsse erkannt, als sie noch gar nicht stattgefunden haben. Das resultierte darin, dass fast alle Messungen falsch waren. Aus diesem Grund wird die Ballposition in den ersten Bildern, bevor der Schuss stattfindet, durch einen Tracking-Algorithmus verfolgt. Dieser Algorithmus verschiebt den Anfangsbereich (in Abbildung 7.16 grau), in dem sich der Ball befindet, sodass dieser immer über den Ball bleibt (blau). Die Idee ist, dass das Differenzbild, das zur Detektion des Schussbeginns verwendet wird, aus dem Ausschnitt mit dem Ball aus dem letzten Bild und dem aktuellen Ausschnitt (nach Tracking) berechnet wird.

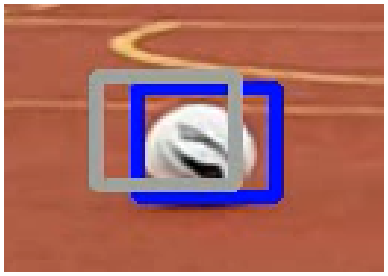

Abbildung 7.16: Funktion Tracking-Algorithmus

OpenCv bietet gleich 8 mitgelieferte Object-Tracker, die für solche Zwecke genutzt werden können. Bei den Trackern handelt es sich um Klassen, die zunächst einmal zu initialisieren sind. Dies geschieht, indem das erste Bild aus dem Video und der zu trackende Bereich in Form der 4 Koordinaten der entsprechenden Methode übergeben werden. Wird der Tracker durch Übergabe des nächsten Bildes aktualisiert, so liefert dieser die aktuellen Koordinaten zurück. Die Tracker unterscheiden sich hinsichtlich Performance und Genauigkeit. Das Problem dieser Tracker ist zum einen, dass sie die Größe des Bildausschnittes verändern, was die Bildung des Differenzbildes erschwert. Zum anderen handelt es sich, wie bereits erwähnt, um Klassen. Diese Klassen können nach Aufruf der nativen Funktionen weder zurückgegeben noch gespeichert werden, was bedeutet, dass sie bei jedem Aufruf neu initialisiert werden müssten. Aus diesen Gründen ist eine eigene Tracking-Funktion zu entwickeln.

Die Idee hinter diesem Tracking-Algorithmus basiert grundsätzlich auf den Differenzbildern. Es wird der Bildausschnitt aus dem vorherigen Bild als Grundlage genommen. Dieser wird innerhalb eines größeren Ausschnittes des neuen Bildes mit allen möglich wählbaren gleich großen Ausschnitten verglichen. Dies geschieht über die Bildung des Differenzbildes der Ausschnitte und der Berechnung des Durchschnittswertes dieser Differenzbilder. Diese Werte werden in einem zweidimensionalen Array, (nachfolgend als Matrix bezeichnet), gespeichert. Abbildung 7.17 verdeutlicht dies.

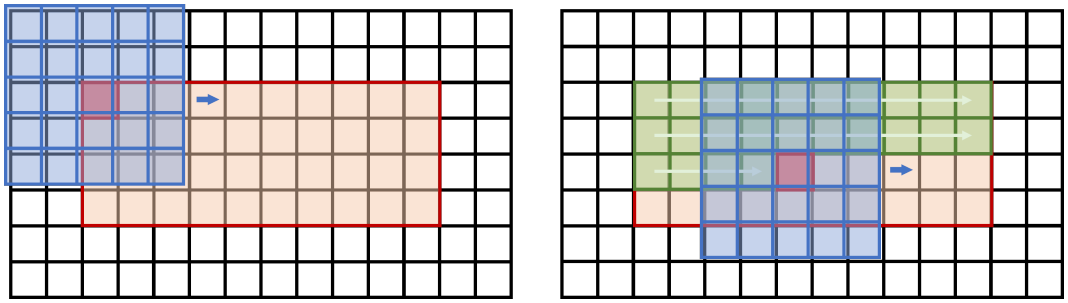

Abbildung 7.17: Funktionsweise des Tracking Algorithmus

Es wird der blau dargestellte Ausschnitt aus dem letzten Bild mit Ausschnitten aus dem neuen Bild (schwarz, im Hintergrund) abgeglichen. Der dabei entstehende Wert wird an die entsprechende Stelle in der Matrix (rot) gespeichert. Dem blauen Pfeil folgend wird diese Berechnung, je um eine Pixel-Reihe verschoben, für jedes Element der Matrix durchgeführt.

Ist dies für jedes Element der Matrix geschehen, so wird der kleinste Wert in dieser Matrix gesucht. Das entsprechende Pixel dazu im Bild wird als neuer Mittelpunkt des Balles erachtet. Es ist daher bei der Initialisierung darauf zu achten, dass die Ball-Box sowohl vertikal, als auch horizontal eine ungerade Anzahl an Pixeln aufweist[8].

Es stellt sich die Frage, wie groß der Bildbereich, in dem der Abgleich stattfindet, zu wählen ist. Um einen großzügigen Bereich zu wählen, der die gesuchte Koordinate enthalten muss, wird die dreifache Größe sowohl vertikal als auch horizontal veranschlagt. Die Auswertung des Algorithmus hat eine überwältigende Trefferquote gezeigt. In 25 Testvideos konnten alle Ballpositionen verfolgt werden, sodass jeder Schuss über das beschriebene Verfahren detektiert werden konnte.

Der Algorithmus in der eben beschriebenen Form ist jedoch viel zu langsam, um eingesetzt werden zu können. Aus diesem Grund wird dieser, weiter optimiert. Abbildung 7.18 zeigt zwei der Matrizen, die die Werte der Abgleichung enthalten.

[8] Bei einer geraden Anzahl kann kein Pixel Mittelpunkt betrachtet werden. Im Test mit nicht in der Hand gehaltenen Videos hat sich gezeigt, dass die „Ball-Box“ die Tendenz hat nach rechts oben zu wandern.

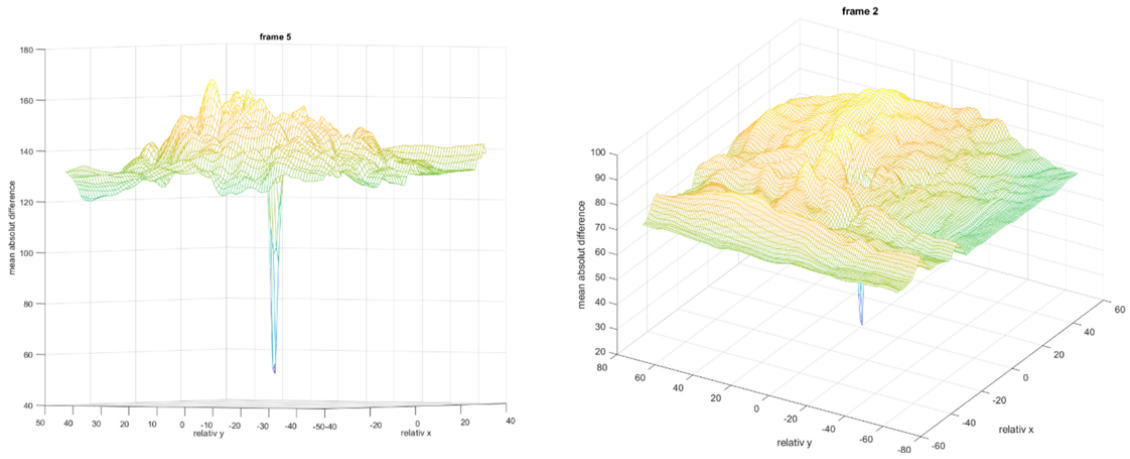


Abbildung 7.18: Darstellung der Abweichungen

Die Ergebnisse zeigen, in welchen Regionen es zu welchen Abweichungen kommt. Die einzelnen Abweichungen befinden sich zum Großteil in einem gewissen Bereich. Doch genau an dem Punkt, wo die Bilder optimal übereinander liegen, kommt es zu einer extrem kleinen Abweichung. Diese Beobachtung konnte in sämtlichen der 2000 betrachteten Matrizen gemacht werden. Einzige Ausnahme bildeten die Bilder, in denen der Schuss erfolgt.

Festzuhalten ist, dass der Punkt, an dem die Bilder bestmöglich übereinander liegen, sehr nah an der Mitte liegt. In den meisten Fällen sind es weniger als 5 Pixel relativ zur Position im letzten Bild, seltener bis zu maximal 10. Aus diesem Grund wird ein neuer Algorithmus entworfen, der bei der Suche nach der besten Überlappung schneller bei Punkten in der Mitte ankommt, ohne einfach den Suchbereich zu verkleinern. Es wird ein Algorithmus entworfen, der vom Mittelpunkt aus nach außen vorgeht und bei Erreichen eines kritischen Punktes den Abgleich abbricht. Abbildung 7.19 zeigt zwei mögliche Algorithmen, nach denen die Auswahl der Pixel erfolgen kann.

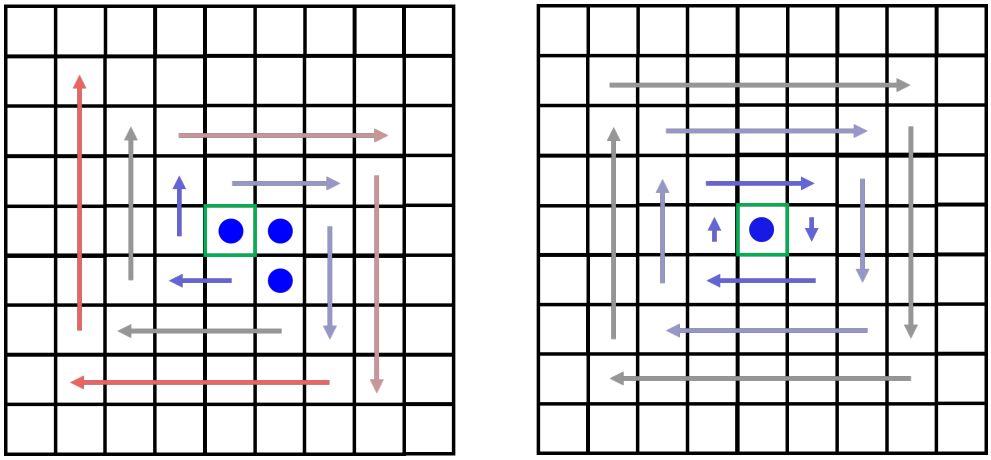

Abbildung 7.19: Vorgehen der Tracking Algorithmen

Das mittlere grüne Feld symbolisiert die Koordinate, an der sich im letzten Bild der Mittelpunkt des Balles befunden hat. Die blauen Punkte entsprechen dabei den Koordinaten, die zu Anfang einzeln ausgewertet werden müssen. Die Felder unter den Pfeilen werden dann in einer fortlaufenden Schleife abgearbeitet. Pfeile gleicher Farbe werden dabei im selben Schleifendurchgang bearbeitet.

Beide Algorithmen werden im Folgenden auf ihre Effizienz und Genauigkeit getestet. Um eine Vergleichbarkeit zu schaffen, wird dazu auch die ursprüngliche Implementierung herangezogen. Diese wird nachfolgend als Algorithmus 3 bezeichnet. Der, in Abbildung 7.19 links zu sehende Algorithmus wird folgend als Algorithmus 1, der rechts abgebildete als Algorithmus 2 bezeichnet. Zum Erstellen der nachfolgenden Ergebnisse wurde ein Python Skript programmiert, in dem für ein ausgewähltes Video alle 3 Algorithmen angewandt werden. Dabei wird für die einzelnen Berechnungen die Zeit gemessen. Um die Zeitmessung im statistischen Mittel unabhängig von eventuellen Hitzeproblemen des Rechners zu gestalten, werden die Algorithmen nicht für das ganze Video hintereinander angewandt, sondern für jedes Bild. Die Überprüfung der Trefferquote findet visuell durch im Bild markierte Ausschnitte statt. Die Zeiten werden als Daten abgespeichert, sodass sie in Abbildung 7.20 aufgezeigt werden können.

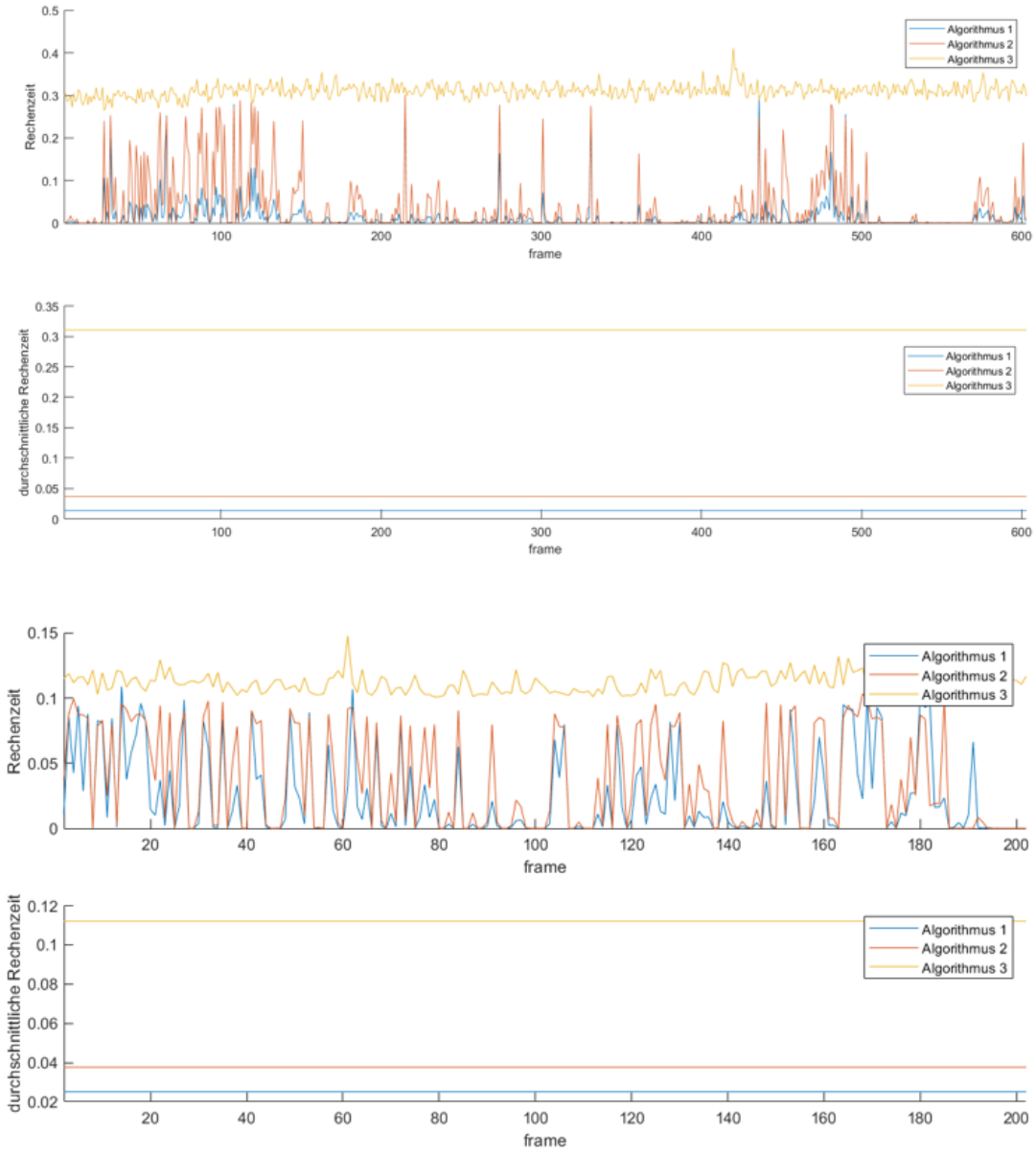


Abbildung 7.20: Rechenzeit der Tracking-Algorithmen

Die erste Abbildung zeigt die Werte eines extra für die Tests der Tracking-Algorithmen angefertigten Videos. Das obere Bild die rohen Messdaten und die des unteren die entsprechenden Mittelwerte. Die zweite Grafik zeigt die gemessenen Werte bei einem der zu analysierenden Schüsse. Wie zu erwarten, liegt die Rechenzeit des dritten Algorith-

mus deutlich über den der beiden Anderen. Bei den andern beiden Algorithmen zeichnet sich eine eindeutige Tendenz ab, dass Algorithmus 1 schneller ist. Um diesen Sachverhalt endgültig zu klären, wird eine weitere Untersuchung vorgenommen. Diese findet in einem imaginärem Grid von 201x201 Pixeln mit dem Mittelpunkt (101,101) statt. Nacheinander wird das Erreichen jeder möglichen Koordinate als Abbruchbedingung genommen. Es wird die Anzahl der Berechnungen gezählt, die für das Überprüfen der Abbruchbedingungen benötigt werden. Um die Algorithmen in Vergleich zu setzen, wurde Abbildung 7.21 erzeugt. In diesem Bild sind die Areale, in denen der Algorithmus 1 weniger Berechnungen braucht schwarz und alle Areale in denen der Algorithmus 2 schneller ist rot gefärbt. Bei gleicher Anzahl ist die Fläche Grün gefärbt. Letzteres trifft nur auf den Mittelpunkt zu.

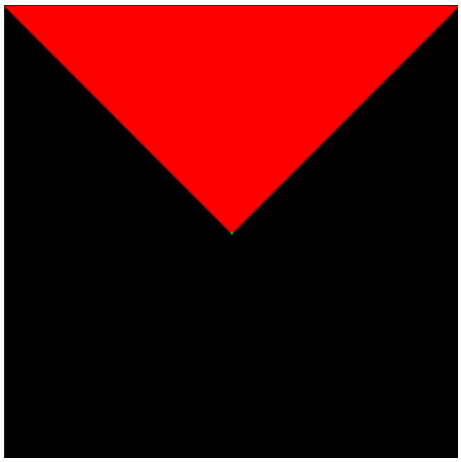

Abbildung 7.21: Bevorzugte Bereiche der Tracking-Algorithmen

Die Abbildung 7.21 zeigt gut, dass die allermeisten Koordinaten schneller vom ersten Algorithmus erreicht werden, als vom zweiten. Dementsprechend ist der erste Algorithmus für das Tracking in 75 Prozent der Fälle schneller. Bezüglich der Genauigkeit haben sich keine großen Unterschiede ergeben. Algorithmus 3 ist im Großen und Ganzen ein wenig genauer, doch alle drei Algorithmen konnten jeweils alle aus der Hand aufgenommenen Videos richtig auswerten. Aufgrund der gewonnenen Erkenntnisse wird für die Implementierung der erste Algorithmus verwendet.

### 7.3.6 Test

Auch für die optimierte Bildverarbeitung werden die Geschwindigkeitsmessungen getestet. Verwendet wurden dieselben Videos, wie beim allgemeinen Algorithmus, sodass Tabelle 7.3 um folgende Werte ergänzt werden kann.

| Video | Radar-Messgerät | allgemeiner Algorithmus | optimierter Algorithmus |
|---|---|---|---|
| 0202 | 55 | 67,8 | 59,6 |
| 0203 | 54 | 59,3 | 64,4 |
| 0204 | 44 | 58,2 | 50,4 |
| 0205 | 47 | 52,8 | 55,0 |
| 0210 | 52 | 64,0 | 61,0 |

Tabelle 7.5: Testergebnisse optimierter Algorithmus (Messungen in km/h)

Die Werte weichen von denen des allgemeinen Algorithmus ab, was zu erwarten war, da die Auswertung durch eine andere Berechnung erfolgt. Es zeichnet sich kein klares Bild ab, ob die Werte tendenziell kleiner oder größer als die des allgemeinen Algorithmus sind. Zu erkennen ist jedoch, dass die Werte näher an denen des allgemeinen Algorithmus sind als bei denen des Radar-Messgerätes. Aus diesem Grund sind die Abweichungen als innerhalb der Toleranz anzusehen.


Abbildung 7.22: Auswertung des Videos 0202.*mp*4

## 7.4 Diskussion der Echtzeitfähigkeit

Abschließend zur Bildverarbeitung ist zu klären, ob diese schnell genug arbeitet, um in Echtzeit implementiert zu werden. Dabei wird die in Kapitel 6.2 beschriebene Definition von Echtzeit zugrunde gelegt. Um einen Eindruck davon zu bekommen, wie lange die Algorithmen brauchen, wird in der Python Implementierung die Zeit gemessen, die die Algorithmen für jeden Durchlauf brauchen. Dazu könne Funktionen aus OpenCV genutzt werden, wie im Quelltext-Auszug 7.14 zu sehen.

```
1 CLOCK_FREQ = cv2.getTickFrequency()
2 t1 = cv2.getTickCount()
3 ...
4 t2 = cv2.getTickCount()
5 processedTime = (t2-t1) / CLOCK_FREQ
```

Listing 7.14: Zeitmessung in Python

Die Ergebnisse wurden auf der Konsole ausgegeben. Beim allgemeinen Algorithmus pendelt die Zeit zwischen 0,02 und 0,05 Sekunden. Damit sind im schlechtesten Fall rein rechnerisch maximal 20 Bilder in der Sekunde realisierbar. Zu diesen 0,05 Sekunde kommen jedoch noch weitere Verzögerungen durch weiteren Programmcode und durch das System, welches im Hintergrund die Bilddaten lädt. Des Weiteren ist noch zu bedenken, dass die Rechenleistung gängiger Smartphones geringer ist, als die des verwendeten Laptops. Alles in einem kann gesagt werden, dass der allgemeine Algorithmus nicht geeignet ist, um die geforderte Bildrate von 24 Bildern pro Sekunde zu realisieren.
Beim optimierten Algorithmus muss zwischen dem Tracking und der Verfolgung der Ballpositionen unterschieden werden. Bei der Verfolgung der Ballpositionen beträgt die benötigte Zeit maximal 0,00002 Sekunden, was eine Auswertung in Echtzeit ermöglichen würde. Das Problem beim optimierten Algorithmus liegt beim Tracking Algorithmus. Die Zeit die dieser benötigt, ist sehr variable und abhängig davon, wie stark sich das Smartphone bewegt hat. Dabei kann dieser bis zu einer Zehntelsekunde brauchen. Aus diesem Grund kann auch der optimierte Algorithmus nicht in Echtzeit implementiert werden. Die App ist so auszulegen, dass die Bilddaten aufgezeichnet, und nach der Aufzeichnung ausgewertet werden.

# 8 Implementierung der App

Der Prototyp der App wird dem Konzept entsprechend implementiert. Es wird dabei ein eigenes Design entwickelt. Als Hintergrundfarbe wurde Schwarz für den dunklen Modus[1] und Weiß für den Normalen verwendet. Als Akzentfarben für Buttons und Anzeigen werden ein helles Türkis und ein kräftiges Blau verwendet[2]. Diese Farben schaffen eine freundlich und frisch wirkende Benutzeroberfläche[3] und setzen sich sowohl von Weiß als auch von Schwarz gut ab. Letzteres ist vor allem bei der Nutzung im Freien wichtig.

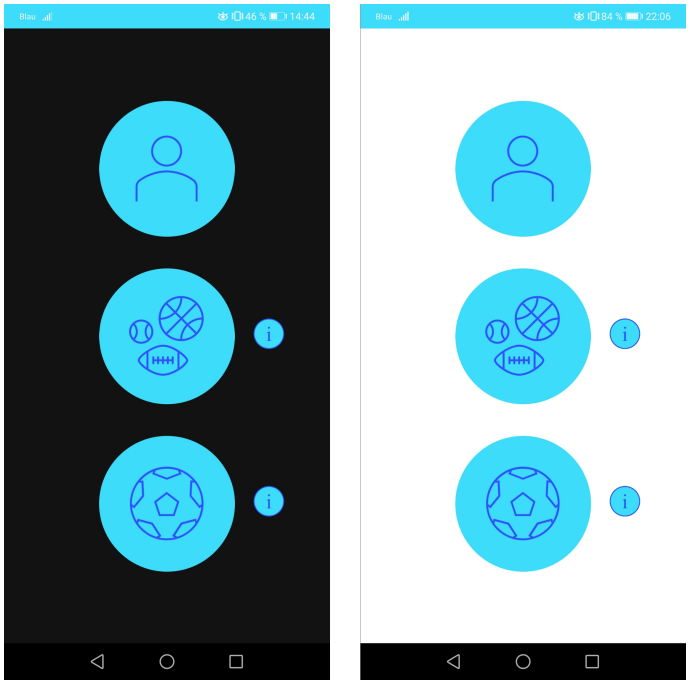

Abbildung 8.1: Design des Hauptmenüs

[1] So wird bei Smartphones die Einstellung bezeichnet, bei der Hintergründe auf dem Smartphone überwiegend in dunklen Farben dargestellt werden.
[2] Es wurden folgende Farben verwendet (r,g,b): Türkis = (60,220,250); Blau = (47,82,255)
[3] Quelle: https://www.gesundheit.de/wellness/sanfte-medizin/weitere-therapien-und-behandlungen/farben-und-ihre-wirkung [Letzter Zugriff: 27.02.2021]

Um zudem die Anforderung N4 einzubinden, werden verstärkt Bilder und Grafiken als Buttons und zu Informationszwecken benutzt. Abbildung 8.1 zeigt das Hauptmenü im dunklen und im normalen Modus. Zudem wird ein Icon[4] für die App entwickelt, wie in Abbildung 8.2 zu sehen ist. In diesem finden sich die Akzentfarben wieder und es zeigt Grafiken, die die Funktionalität dieser App zusammenfassen.

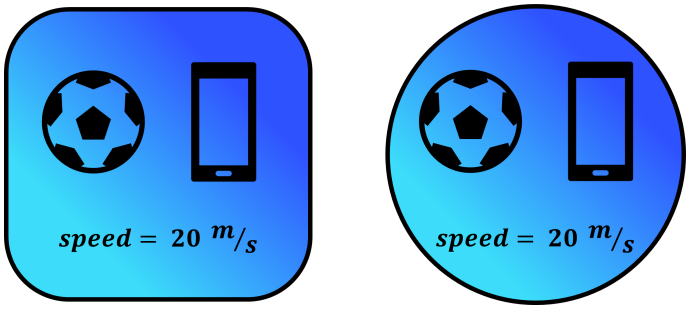


Abbildung 8.2: Icon und rundes Icon der App

Die Icons werden in Microsoft Word erzeugt. Dort können mit Formen und, von Microsoft Word bereitgestellten, Piktogrammen diese Grafiken erzeugt und als Bilder gespeichert werden. Diese Bilder können dann dem Android Projekt hinzugefügt und genutzt werden.

Die App wurde mit dem nativen Typ erstellt, wie in den Grundlagen (Kapitel 2.1.2) beschrieben ist. Zudem wurde OpenCV als Modul hinzugefügt, damit die entsprechenden Funktionen und Klassen genutzt werden können. Im Folgenden wird die Implementierung der wichtigsten Teile der App beschrieben. Eine wichtige Rolle bei der Implementierung spielt auch die Kameravorschau für die Bildverarbeitung. Wie in Kapitel 6.2 beschrieben, gibt es für die Kameravorschau zwei Activities; eine für die allgemeine und eine für die optimierte Bildverarbeitung. Diese werden im Folgenden durch die Begriffe „allgemeine“ und „optimierte Activity“ referenziert.

[4] In Android gibt es für eine App zwei Icons. Je nach Smartphone-Hersteller werden eher eckige oder runde Icons verwendet.

## 8.1 Hauptmenü

Das Hauptmenü ist die Main Activity dieser App, sprich die Activity, mit der die App startet. Wie im Konzept (Kapitel 6.2) beschrieben und in Abbildung 8.1 zu sehen ist, besteht diese aus fünf Buttons. Einen für die Spielerverwaltung (oben) und einen für je die allgemeine (Mitte) und die optimierte Bildverarbeitung (unten). Zu den beiden Letzteren gibt es zusätzlich noch einen Informationsbutton. Über die Buttons gelangt der Nutzer zu den entsprechenden Activities.

Die Buttons werden durch „Image Views“ realisiert, denen in der Java-Datei „onClick Listenern“ hinzugefügt werden. Der Quelltext-Auszug 8.1 zeigt den Ausschnitt der Layout-Datei, in der das Bild für den Fußball eingefügt wird (in Abbildung 8.1 der unterste Button).

```
1  <ImageView
2          android:id="@+id/ivMainBtnFball"
3          android:layout_width="150dp"
4          android:layout_height="150dp"
5          android:layout_marginTop="35dp"
6          app:layout_constraintEnd_toEndOf="parent"
7          app:layout_constraintStart_toStartOf="parent"
8          app:layout_constraintTop_toBottomOf="@+id/ivMainBtnAball"
9          app:srcCompat="@drawable/mainbtnfball" />
```

Listing 8.1: Image View in der Layout-Datei

Über die „findViewById“ Funktion wird das „Image View“ in Java referenziert und diesem der Listener hinzugefügt. Wird auf das „Image Views“ geklickt, so wird eine neue Activity aufgerufen, wie in den Zeilen 5 bis 7 des Auszuges 8.2 zu sehen ist. Aufgerufen wird die „AskPermissionActivity“, die in Kapitel 8.2 näher erklärt wird.

```
1  ivbtnFBall = findViewById(R.id.ivMainBtnFball);
2  ivbtnFBall.setOnClickListener(new View.OnClickListener() {
3      @Override
4      public void onClick(View view) {
5          Intent intent = new Intent(getApplicationContext(),
6                                     AskPermissionActivity.class);
7          intent.putExtra("target", "optimiert");
8          startActivity(intent);
9      }
10 });
```

Listing 8.2: Image View in der Java-Datei

Für diese Activity wird die „onBackPressed-Methode“ überschrieben. Diese beschreibt, was passiert, wenn die Zurück-Taste (dreieckiges Symbol in Abbildung 8.3) gedrückt wird. Damit wird verhindert, dass die App durch versehentliche Berührungen zu schnell beendet wird.

```
1  @Override
2  public void onBackPressed(){
3       if(allowExit){
4            this.finishAffinity();
5       } else {
6            allowExit = true;
7            Toast.makeText(this, "Nochmal druecken, um die App zu beenden.",
8                 Toast.LENGTH_SHORT).show();
9
10           new Handler().postDelayed(new Runnable() {
11                @Override
12                public void run() {
13                allowExit = false;
14                }
15           }, 3000);
16      }
17 }
```

Listing 8.3: onBackPressed-Methode der Main Activity

Wird die Zurück-Taste gedrückt, wird die boolsche Variable „allowExit“ abgefragt. Ist diese logisch falsch, so wird sie auf logisch wahr gesetzt. Anschließend erfolgt über einen Toast[5] die Information an den Nutzer, dass dieser die Taste erneut drücken muss, um die App zu beenden (siehe Abbildung 8.3). Danach wird ein Handler initialisiert, der im Hintergrund nach einer Zeit von drei Sekunden die Variable wieder auf logisch falsch setzt. Ist die Variable beim Drücken der Zurück-Taste auf logisch wahr gesetzt, so wird die App mit „this.finishAffinity()“ beendet.

[5]Kleiner Pop-up-Text zur Information des Nutzers, siehe Abbildung 8.3.

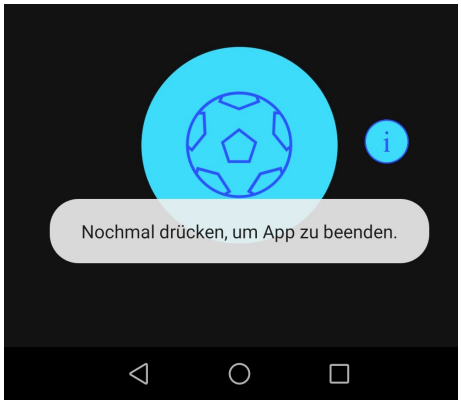


Abbildung 8.3: Info zum Beenden der App

## 8.2 Erteilen der Berechtigungen

Damit eine App in Android bestimmte Features des Smartphones nutzen darf, muss der Nutzer der App erst die benötigten Berechtigungen geben. Welche Berechtigungen eine App benötigt, muss im Manifest der App aufgelistet werden. Bei der Berechtigung, die diese App braucht, handelt es sich um den Zugriff auf die Kamera, wie im Auszug 8.4 aus dem Manifest der App zu sehen ist.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3      package="com.example.ballgeschwindigkeitsmesser">
4
5      <uses-permission android:name="android.permission.CAMERA" />
6      <uses-feature android:name="android.hardware.camera" />
7
8      ...
9
10 </manifest>
```

Listing 8.4: Berechtigungen im Mainfest der App

Damit der Nutzer diese Berechtigungen nicht umständlich in den Smartphone-Einstellungen vornehmen muss, ist es aus Gründen der Nutzerfreundlichkeit besser, wenn die App selber nach den benötigten Berechtigungen fragt. Dies wird in dieser App durch die Activity „AskPermissionActivity“ vorgenommen. Diese wird sowohl vor der Activity für die allgemeine Bildverarbeitung, als auch vor der für die Optimierte aufgerufen. Zunächst einmal

wird überprüft, ob die Berechtigungen bereits erteilt sind, wie in Quelltext-Auszug 8.5 zu sehen ist.

```
1  if (ContextCompat.checkSelfPermission(this, Manifest.permission.CAMERA) ==
2       PackageManager.PERMISSION_GRANTED && ContextCompat.checkSelfPermission(
3       this, Manifest.permission.WRITE_EXTERNAL_STORAGE) ==
4       PackageManager.PERMISSION_GRANTED && ContextCompat.checkSelfPermission(
5       this, Manifest.permission.READ_EXTERNAL_STORAGE) ==
6       PackageManager.PERMISSION_GRANTED) {
7            onPermissionGranted();
8  } else {
9       btn.setOnClickListener(new View.OnClickListener() {
10           @Override
11           public void onClick(View view) {
12                requestPermissions(new String[] {
13                     Manifest.permission.CAMERA },
14                     REQUEST_CODE_PERMISSIONS);
15                }
16            });
17 }
```

Listing 8.5: Berechtigungen überprüfen

Sind die Berechtigungen erteilt, so wird der Nutzer direkt weitergeleitet[6]. Sind die Berechtigungen nicht erteilt, so werden diese vom Benutzer erbeten. Durch Klick auf den Button wird die in Android implementierte Frage nach den Berechtigungen aufgerufen. Dazu erscheinen dem Nutzer für die entsprechenden Berechtigungen Pop-up-Fenster vom System, in denen er die Berechtigungen erteilen oder verweigern kann. Ist dies geschehen, wird die Methode „onRequestPermissionsResult“ aufgerufen, der die Ergebnisse der Nutzerabfrage übergeben werden.

```
1  @Override
2  public void onRequestPermissionsResult(int requestCode, String[] permissions,
3       int[] grantResults) {
4       switch (requestCode) {
5            case REQUEST_CODE_PERMISSIONS:
6                 if(grantResults.length > 0){
7                      for(int i = 0; i < grantResults.length; i++){
8                      if(grantResults[i] !=
9                           PackageManager.PERMISSION_GRANTED){
10                     onPermissionDenied();
11                     return;
12                     }
```

[6]Zu welcher Activity der Nutzer weitergeleitet wird, hängt davon ab welches Aufrufargument der Activity mitgegeben wird. Im Listing 8.2 ist dies in Zeile 6 zu sehen.

```
13                                      }
14                                      onPermissionGranted();
15                                  } else {
16                                      onPermissionDenied();
17                                  }
18                                  break;
19                              default: onPermissionDenied();
20                      }
21      }
```

Listing 8.6: Ergebnis der Berechtigungerteilung verarbeiten

Hat der Nutzer alle Berechtigungen erteilt, so wird er weitergeleitet. Hat er nicht alle nötigen Berechtigungen erteilt, so wird er über die zentrale Textausgabe der Activity gebeten dies zu tun. Der Nutzer hat dann die Möglichkeit den entsprechenden Button erneut zu drücken.

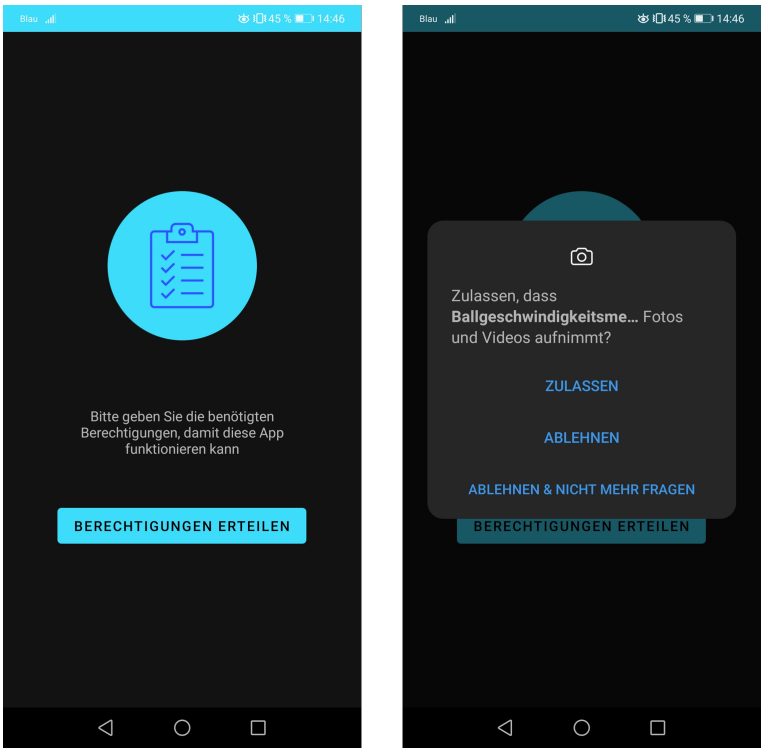


Abbildung 8.4: Berechtigungen erteilen

Die Berechtigungen auf diese Weise zu erteilen, ist um einiges benutzerfreundlicher, als dies über die Einstellungen vorzunehmen. So werden die Anforderungen nach einer benutzerfreundlichen App, die intuitiv zu bedienen ist, mit eingebunden.

## 8.3 Allgemeine Bildverarbeitung

Wie im Konzept geplant, wird für die Implementierung das OpenCV-Interface verwendet. Dieses wird, wie in den Grundlagen im Kapitel 2.1.1 beschrieben, implementiert. Dabei wird die Version 3.4.2 verwendet[7]. Der Quelltext-Auszug 8.7 zeigt die Implementierung der Kameravorschau in der XML-Datei.

```
1  <androidx.constraintlayout.widget.ConstraintLayout
2        android:id="@+id/myconstraintLay"
3        android:layout_width="match_parent"
4        android:layout_height="match_parent"
5        app:layout_constraintBottom_toBottomOf="parent"
6        app:layout_constraintEnd_toEndOf="parent"
7        app:layout_constraintStart_toStartOf="parent"
8        app:layout_constraintTop_toTopOf="parent">
9
10          <org.opencv.android.JavaCameraView
11             android:id="@+id/mycamview"
12             android:layout_width="match_parent"
13             android:layout_height="match_parent"
14             app:layout_constraintBottom_toBottomOf="parent"
15             app:layout_constraintEnd_toEndOf="parent"
16             app:layout_constraintStart_toStartOf="parent"
17             app:layout_constraintTop_toTopOf="parent" />
18
19 </androidx.constraintlayout.widget.ConstraintLayout>
```

Listing 8.7: OpenCV Kameravorschau

Die Angaben „match_ parent“ sorgt dafür, dass die Kameravorschau den verfügbaren Platz auf dem Bildschirm ausnutzt. Die Abbildung 8.5 zeigt die allgemeine Activity.

[7] In vorangegangenen Tests hat sich diese Version als geeignet erwiesen, da sie über alle benötigten Funktionalitäten verfügt. Des Weiteren sind zu dieser Version mehrere Tutorials und Dokumentationen online verfügbar. Neuere Versionen sind abweichend, sodass sie nicht wie gewünscht implementiert werden konnten.

Abbildung 8.5: Design der Activity der allgemeinen Bildverarbeitung

Die Activiy ist für den Landscape-Mode ausgelegt. Um das Bild optimal auszunutzen wird zudem der Fullscreen-Mode eingestellt, bei dem sämtliche Bedienelemente wie die Softkeys oder die Benachrichtigungsliste ausgeblendet werden. Das türkise Feld in der Mitte zeigt dem Nutzer den Bereich, der im Algorithmus für die Detektion der Ballpositionen verwendet wird. Auf der rechten Seite befindet sich der Aufnahmebutton, mit dem die Aufnahme gestartet und gestoppt werden kann. Zudem befindet sich dort ein Button für die Einstellungen, über den die Distanz und der Kamerawinkel eingegeben werden können, wie im Konzept (Kapitel 6.2) erwähnt.

Unter dem Aufnahmebutton befindet sich ein Button mit einem Pfeil. Wird auf diesen gedrückt, so wechseln dieser und der Aufnahmebutton die Seite. Erfolgt der Wurf nicht von links nach rechts, sondern von rechts nach links, so kann es dazu kommen, das der Aufnahmebutton die Sicht auf den Spieler verdeckt. Das kann mit diesem Button umgangen werden, wie im Abbildung 8.6 zu sehen ist. Diese Funktion bietet dem Nutzer mehr Flexibilität und erhöht die Nutzerfreundlichkeit.

Abbildung 8.6: Verschieben des Aufnahmebuttons

Um die Anzeige der Softkeys oder der Benachrichtigungsliste zu kontrollieren, implementiert die Activity das Interface „View.OnSystemUiVisibilityChangeListener“. Dieses Interface benachrichtigt die Activity jedes Mal, sobald sich die Sichtbarkeit der Bedienelemente ändert, indem die Methode „onSystemUiVisibilityChange“ aufgerufen wird. In diesem Fall wird Quelltext 8.8 ausgeführt.

```
1  @Override
2  public void onSystemUiVisibilityChange(int visibility) {
3       if ((visibility & View.SYSTEM_UI_FLAG_FULLSCREEN) == 0) {
4            new Handler().postDelayed(new Runnable() {
5                      @Override
6                      public void run() {
7                           hideSystemUI();
8                      }
9            }, 3000);
10      }
11 }
```

Listing 8.8: Änderung der Sichtbarkeit der Bedienelemente

Mit der Funktion „hideSystemUI“ wird der Fullscreen Mode eingestellt. Auch hier wird ein Handler im Hintergrund verwendet, der nach drei Sekunden den Fullscreen Mode wiederherstellt. Dieser kann unter anderem dann beendet werden, wenn der Nutzer vom oberen Bildrand nach unten wischt, um die Benachrichtigungsliste einzublenden. Dass sich der Fullscreen Mode automatisch wieder herstellt, dient der Benutzerfreundlichkeit. Auch in dieser Activity wird die Funktion der Zurück-Taste überschrieben. sodass auch hier wird die Funktion erst nach zweimaligem Klicken aktiviert. Um danach nicht zur

Activity mit der Berechtigungsabfrage zurückzukehren[8], wird durch diese Funktion die Activity des Hauptmenüs neu aufgerufen.

Durch Klicken auf das Zahnrad unten rechts in der Activity, kann der Nutzer die Werte für die Entfernung zur Fluglinie und den Kamerawinkel anpassen. Die Eingabe dieser Werte erfolgt durch ein Pop-up-Fenster.

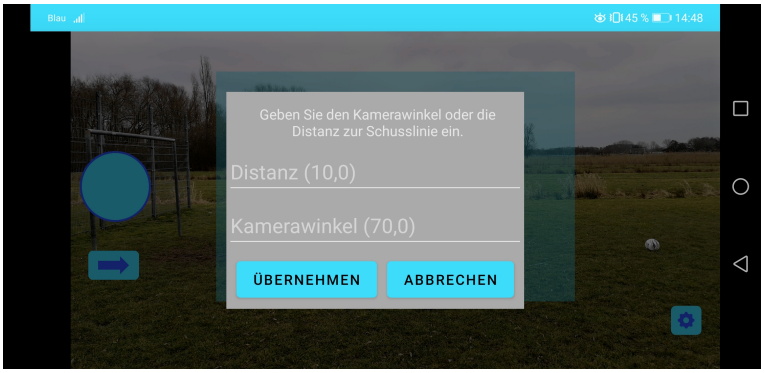


Abbildung 8.7: Eingabe der Parameter

Um es dem Nutzer einfacher zu machen, werden die aktuellen Werte als Hinweis in den entsprechenden Textfeldern angezeigt. Nach Klicken auf den Übernehmen-Button werden beide Textfelder überprüft, ob es eine Eingabe gab. Ist dies der Fall, so werden die Werte übernommen und sofort im Langzeitspeicher gespeichert, damit diese beim nächsten Aufruf der App wieder verwendet werden können[9].

```
1  SharedPreferences mySavedDistance = getSharedPreferences("mySavedDistance", 0);
2  SharedPreferences.Editor myEditor = mySavedDistance.edit();
3
4  try {
5        DISTANCE_ALG = Double.parseDouble( etDistanzAlg.getText().toString() );
6        myEditor.putString("DISTANCE_ALG", String.format("%f", DISTANCE_ALG));
7  } catch (Exception e){
8        ...
```

[8] Durch die Zurück-Taste gelangt der Nutzer normalerweise zur vorherigen Activity zurück. In diesem Fall macht dies aber kein Sinn, da dies die Activity mit der Berechtigungsabfrage ist. Diese Activity bietet keinen Nutzen für den Nutzer und ist aus diesem Grund aller Wahrscheinlichkeit nach nicht das Ziel der Aktion des Nutzers.

[9] Der Quelltext-Auszug 8.9 ist verkürzt dargestellt. Nachdem die Werte aus dem Textfeld gelesen wurden, findet eine Überprüfung statt, ob die Werte innerhalb eines gültigen Bereiches liegen. Für den Kamerawinkel ist das ein Bereich zwischen, einschließlich 45 und 90 Grad und für die Entfernung zur Fluglinie 2 bis 20 Meter.

```
9   }
10
11  try {
12        ANGLE = Double.parseDouble( etAngle.getText().toString() );
13        myEditor.putString("ANGLE", String.format("%f", ANGLE));
14  } catch (Exception e){
15        ...
16  }
17
18  ...
19
20  myEditor.commit();
21  popupWindow.dismiss();
```

Listing 8.9: Werte übernehmen

Die Daten werden über „SharedPreferences“ gespeichert[10], damit schnell und unkompliziert auf sie zugegriffen werden kann. Nach Eingabe der Werte werden diese zur Information des Nutzers erneut über einen Toast ausgegeben, damit dieser unkompliziert sicherstellen kann, dass die Werte auch übernommen werden.

Nach einem Klick auf den Aufnahmebutton wird die Aufnahme der Bildsequenz gestartet. Dies wird dem Nutzer durch die Zeichensequenz „REC“ in der oberen linken Bildecke und durch eine Änderung des Aufnahmebuttons angezeigt.


Abbildung 8.8: Design der Aufnahme

[10]SharedPreferences ist eine API in Android, mit der verhältnismäßig kleine Datenmengen gespeichert werden können. Siehe https://developer.android.com/reference/android/content/SharedPreferences [Letzter Zugriff: 05.03.2021] oder: https://developer.android.com/training/data-storage/shared-preferences [Letzter Zugriff: 05.03.2021],

Die Bilddaten werden gespeichert, indem beim Aufrufen der Methode „onCameraFrame“ die übergebenen Bilddaten in einer Array-Liste gespeichert werden. Der Quelltext-Auszug 8.10 zeigt die Deklaration und Initialisierung der Array-Liste.

```
1  ArrayList<CameraBridgeViewBase.CvCameraViewFrame> frameArrayList =
2        new ArrayList<CameraBridgeViewBase.CvCameraViewFrame>();
```

Listing 8.10: Deklaration und Initialisierung der Array-Liste

Bei einem zuvor einzustellenden Maximum an Bilddaten bricht die Aufnahme automatisch ab. Dies wird dem Nutzer angezeigt, indem die Zeichensequenz „REC“ automatisch verschwindet. Das Maximum an Bilddaten hängt unter anderem vom Arbeitsspeicher des verwendeten Smartphones ab. Ein genauer Zahlenwert ist bei der Weiterentwicklung der App zu ermitteln. Für den Prototyp wurden pauschal 100 Bilder gewählt.

```
1  @Override
2  public Mat onCameraFrame(CameraBridgeViewBase.CvCameraViewFrame inputFrame) {
3        if(isRec){
4              frameArrayList.add(inputFrame);
5              if (frameArrayList.size() >= 100){
6                    isRec = false;
7                          isOverfull = true;
8                          ivRecInfo.setVisibility(View.INVISIBLE);
9                    }
10           }
11           return inputFrame.rgba();
12 }
```

Listing 8.11: onCameraFrame Methode mit Aufzeichnung

Wird nach laufender Aufnahme erneut auf die Aufnahmetaste geklickt, wird die Aufnahme beendet und es erscheint ein Pop-up-Fenster, dass den Nutzer fragt, ob dieser die Aufnahme analysieren oder verwerfen will.


Abbildung 8.9: Pop-up-Fenster zur Auswertung der Daten

Wird die Aufnahme verworfen, so wird die Array-Liste mit dem Befehl „clear“ geleert. Die Analyse der Aufnahme ist im Zuge dieser Arbeit nicht implementiert worden, da es sich lediglich um den Prototyp handelt. In diesen Fall haben beide Button die gleiche Funktion. Die weitere Analyse der Daten ist bei einer Weiterentwicklung der App zu implementieren.

## 8.4 Optimierte Bildverarbeitung

Die Activity für die optimierte Bildverarbeitung ist der Activity, für die allgemeine, ähnlich. Beispielsweise ist die Kameravorschau auf die gleiche Weise implementiert. Auch das Aufnehmen der Bilddaten gleicht der, in der allgemeine Activity. Im Gegensatz zur allgemeinen Activity befindet sich in der Kameravorschau allerdings nicht das türkise Feld, sondern grafische Elemente, mit denen der Nutzer die drei einzugebenden Positionen markieren kann.

Abbildung 8.10: Design der Activity der optimierten Bildverarbeitung

Für die Markierung des Balles ist das türkise halb transparente Feld auf der linken Seite der Abbildung 8.10. Die beiden Torpfosten werden mit den blauen Linien markiert. Die Elemente können vom Nutzer beliebig verschoben werden. Um die grafischen Elemente auf dem Bildschirm zeigen zu können, muss im Layout über der „JavaCameraView“ ein „SurfaceView“ eingefügt werden, in das die grafischen Elemente gezeichnet werden können. Dieses muss genau über der „JavaCameraView“ liegen. Zu diesem Zweck wurde, wie in Quelltext-Auszug 8.7 zu sehen, die „JavaCameraView“ in das „ConstraintLayout“ eingebettet. Dieses kann in der Java-Datei referenziert werden, wie in Quelltext-Auszug 8.12 zu sehen ist.

```
1  customSurfaceView = new MyCustomSurfaceView(getApplicationContext());
2  constraintLayout = findViewById(R.id.myconstraintLay);
3  constraintLayout.addView(customSurfaceView);
```

Listing 8.12: SurfaceView hinzufügen

Dieser Auszug zeigt, wie dem „ConstraintLayout“ ein weiteres Element hinzugefügt wird. Dies ist eine selbst erstellte Klasse mit dem Namen „MyCustomSurfaceView“, die von der Klasse „SurfaceView“ erbt und mit der es möglich ist, mit Kästen und Linien grafische Elemente zu erzeugen.

```
1  private class MyCustomSurfaceView extends SurfaceView implements
2       SurfaceHolder.Callback, View.OnTouchListener {
3
4       ...
5
6       public MyCustomSurfaceView(Context context) {
```

```
7               super(context);
8
9               this.setZOrderOnTop(true);
10
11              surfaceHolder = getHolder();
12              surfaceHolder.addCallback(this);
13              this.getHolder().setFormat(PixelFormat.TRANSLUCENT);
14
15              ...
16
17              this.setOnTouchListener(this);
18      }
19  }
```

Listing 8.13: MyCustomSurfaceView

Zunächst einmal wird mit Zeile 8 dafür gesorgt, dass die erstellte Instanz der Klasse im Layout an oberster Stelle liegt, damit alle Elemente zu sehen sind. Dann ist ein Callback zu initialisieren, der zurückmeldet, sobald die Ansicht bereit ist, sodass die grafischen Elemente gezeichnet werden können. Um die Elemente verschieben zu können, muss diese Klasse zusätzlich über einen Touch-Listener verfügen, der Berührungen an die Klasse meldet. Dies ist in Zeile 16 der Fall. Der Auszug 8.14 zeigt die Funktion, mit der die Elemente gezeichnet werden.

```
1   public void drawContent() {
2       canvas = surfaceHolder.lockCanvas();
3
4       canvas.drawColor(Color.TRANSPARENT, PorterDuff.Mode.CLEAR);
5       canvas.drawColor(Color.TRANSPARENT);
6
7       canvas.drawRect(pleft-3, 0, pleft+3, 1080, colorBlue);
8       canvas.drawRect(pright-3,0, pright+3,1080, colorBlue);
9
10      canvas.drawRect(ballx-ballsize,bally-ballsize,ballx+ballsize,
11              bally+ballsize,colorTurquoise);
12
13      surfaceHolder.unlockCanvasAndPost(canvas);
14  }
```

Listing 8.14: Grafische Elemente zeichnen

Sobald die Ansicht bereit ist, wird die Funktion „drawContent“ aufgerufen. In dieser werden zunächst in den Zeilen 5 und 6 alle bereits gezeichneten Elemente gelöscht. Dies ist wichtig, da beim Verschieben der Elemente, diese ständig neu gezeichnet werden. Dann werden die Linien für die Torpfosten und der Kasten für den Ball gezeichnet. Damit

der Nutzer nicht auswählen muss, welches der Elemente er verschieben will, wird beim Berühren des Bildschirmes berechnet, welches der Elemente am nächsten an der berührten Stelle liegt. Dieses wird dann verschoben. Der Auszug 8.15 zeigt den dazugehörigen Quelltext.

```
1  public boolean onTouch(View view, MotionEvent event) {
2
3       float dleft,dright,dball;
4
5          dleft = Math.abs( event.getX() - pleft );
6          dright = Math.abs( event.getX() - pright );
7          dball = (float) Math.sqrt( (ballx - event.getX()) * (ballx - event.getX()) +
8                              (bally - event.getY()) * (bally - event.getY()) );
9
10         if(dleft < dright){
11           if(dleft < dball){
12                      pleft = event.getX();
13                } else {
14                      ballx = event.getX();
15                      bally = event.getY();
16                }
17      } else{
18           if(dright < dball){
19                      pright = event.getX();
20                } else {
21                      ballx = event.getX();
22                      bally = event.getY();
23                }
24         }
25
26      drawContent();
27         return true;
28 }
```

Listing 8.15: Elemente verschieben

Über den Plus- und Minus-Button kann die Größe der Ball-Box in Schritten von 5 Pixeln verändert werden. Bis zu maximal 150 und minimal 5 Pixeln[11]. Auch hier kann, wie bei der allgemeinen Activity, die Aufnahmetaste verschoben werden.

[11] Gemeint ist dabei die Variable „ballsize“, die die Größe vom Mittelpunkt aus angibt.

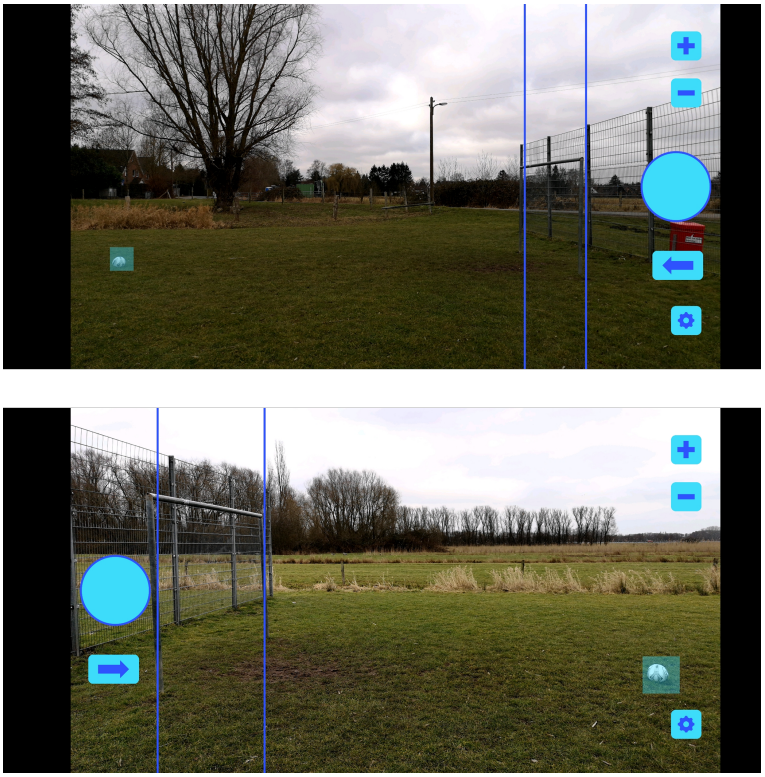

Abbildung 8.11: Verschiedene Größen der Ballboxen

Mit dem Button für die Einstellungen kann hier die Schussweite eingegeben werden. Dabei wird bei der Übernahme der Werte wie bei der allgemeinen Activity vorgegangen.

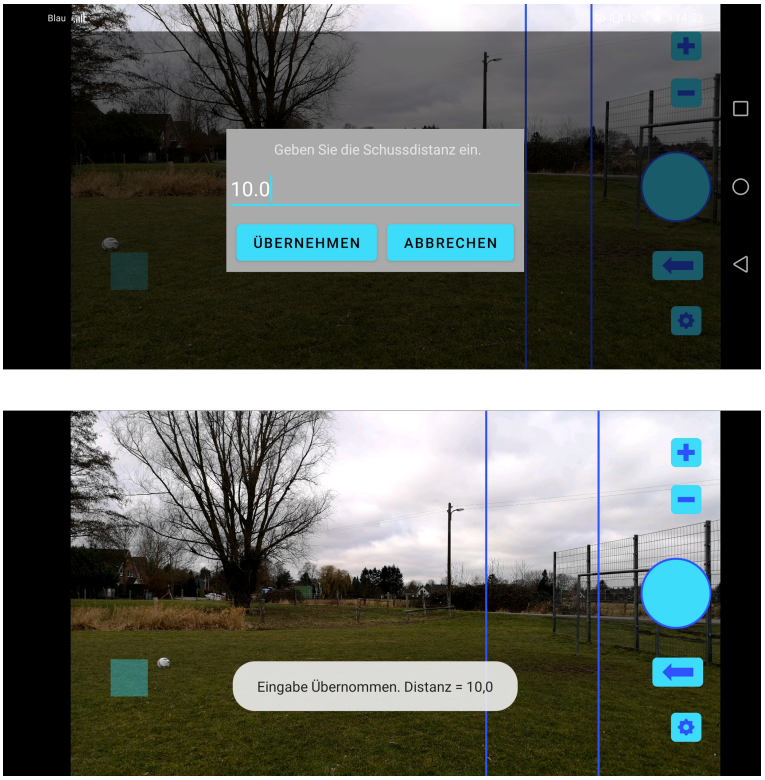


Abbildung 8.12: Änderung der Schussdistanz

Auch in dieser Activity ist die Analyse der aufgenommenen Bildsequenz nicht implementiert.

## 8.5 Info Activity

Da sich der Nutzer erst einmal mit der App vertraut machen muss, ist eine Erklärung zur Nutzung der Activities für die Bildverarbeitung implementiert. Zu dieser gelangt der Nutzer vom Hauptmenü aus über die Buttons mit dem i, was für Info steht. Diese Activities bestehen aus einer Überschrift, die erklärt was mit dieser Anwendung gemacht werden kann und einer „ScrollView“, in der mit Textfeldern und Bildern die Nutzung erklärt wird. Abbildung 8.13 zeigt die Erklärung für die allgemeine Activity.

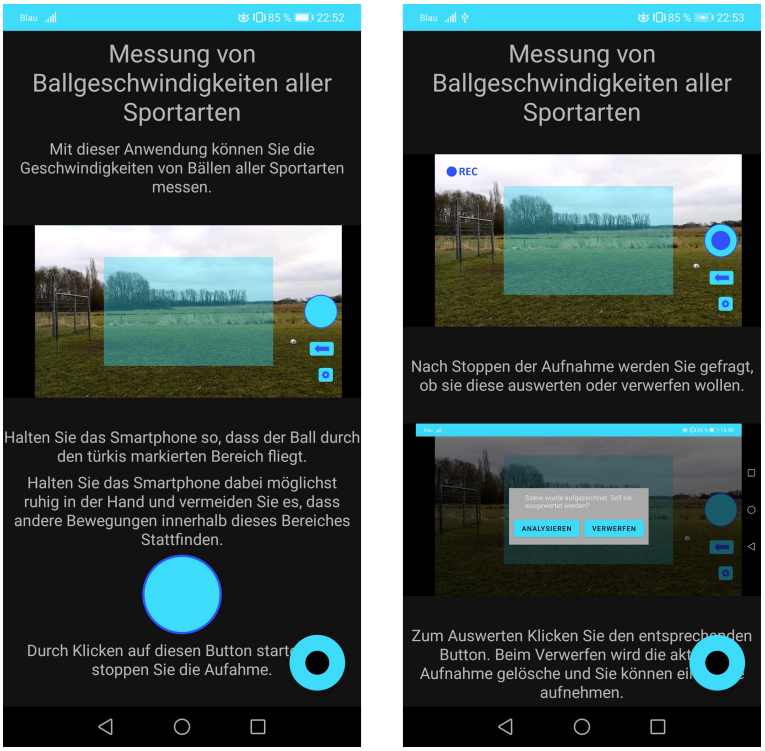


Abbildung 8.13: Erklärung in der Info-Activity

Über den Floating-Action-Button unten links in der Ecke erreicht der Nutzer die Activity der Bildverarbeitung.

## 8.6 Spielerverwaltung

Schlussendlich wird an dieser Stelle noch auf die Spielerverwaltung eingegangen. Diese ist, wie in Abbildung 8.14 zu sehen, designt. Die Spielerverwaltung ist in diesem Prototyp nicht abschließend implementiert. Sie verfügt lediglich über einige grundlegende Eigenschaften.


Abbildung 8.14: Spielerverwaltung

Hauptsächlich besteht die Activity aus einem „ListView“, in dem alle Spieler mit Namen und einigen statistischen Werten angezeigt werden. Neben der gemessenen Durchschnittsgeschwindigkeit und dem Rekordwert wird auch eine Tendenz angezeigt, ob sich die gemessene Geschwindigkeit erhöht oder verringert. Ein „ListView“ bietet die Möglichkeit eine scrollbare Ansicht aller Elemente, etwa einer Array-Liste, zu erzeugen. In diesem Fall wird sie verwendet, um die Liste aller Spieler auszugeben. Intern wird ein Spieler durch die Klasse „Spieler“ repräsentiert.

```
1  public class Spieler {
2
3      private String Name;
4      private int Spielernummer;
5      private String SpielerID;
6
7      private double meanSpeed;
8      private double recordSpeed;
9
```

```
10      private int tendency;
11
12      private final int TENDENCY_INCREASING = 1;
13      private final int TENDENCY_CONSISTENT = 0;
14      private final int TENDENCY_DECREASING = -1;
15
16      public Spieler(String Name, int Spielernummer){
17          this.Name = Name;
18          this.Spielernummer = Spielernummer;
19          this.SpielerID = "Spieler_" + Spielernummer;
20          this.tendency = this.TENDENCY_CONSISTENT;
21          this.meanSpeed = 0.0;
22          this.recordSpeed = 0.0;
23      }
24
25  }
```

Listing 8.16: Klasse Spieler

Damit es zu keinen Verwechslungen kommt, wenn der Nutzer zwei Spieler mit dem gleichen Namen anlegt, erfolgt die Identifikation der einzelnen Spieler intern über eine eindeutige Nummer. Diese wird beim Anlegen für jeden Spieler erzeugt und ist eine Nummer die, beginnend beim Wert 1, hochgezählt wird. Um ein „ListView“ zu implementieren, muss diesem bei der Initialisierung ein sogenannter Adapter hinzugefügt werden.

```
1  spielerListe = findViewById(R.id.listeSpielerverwaltung);
2  SpielerAdapter spielerAdapter = new SpielerAdapter(this, alleSpieler);
3  spielerListe.setAdapter(spielerAdapter);
```

Listing 8.17: ListView initialisieren

Dieser Adapter definiert, wie jedes einzelne Element angezeigt wird. Dazu ist eine weitere Layout-Datei notwendig, die das Aussehen jedes Elementes der Liste definiert, wie in der Abbildung 8.15 zu sehen ist.

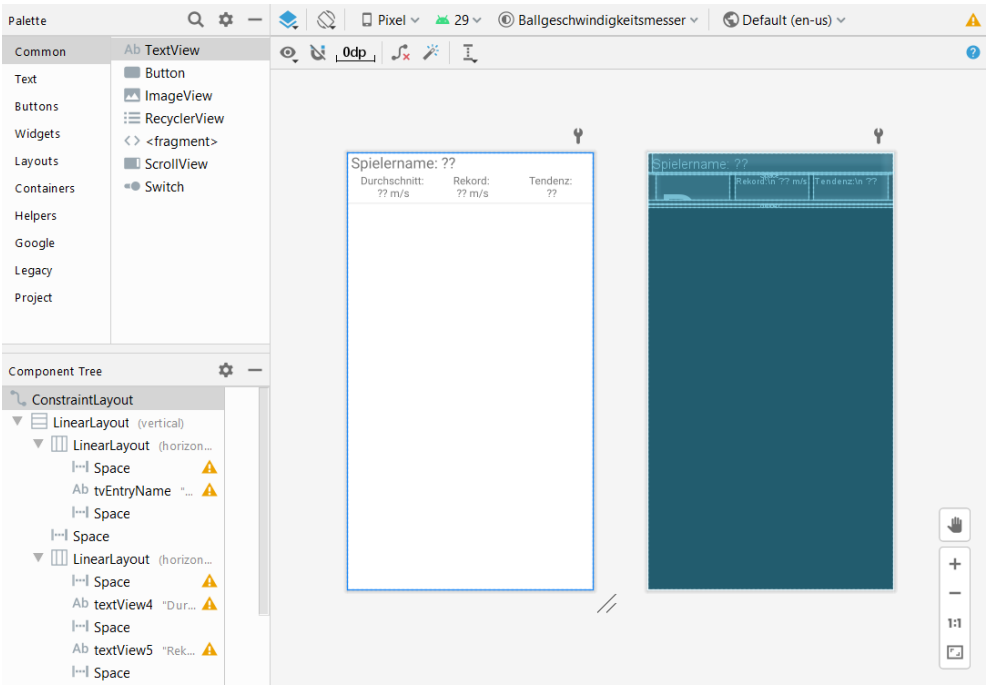


Abbildung 8.15: Design der Spielereinträge

Dieses Layout wird im Adapter für jedes Element geladen und bearbeitet, wie im Quelltext-Auszug 8.18 zu sehen ist.

```
1  @Override
2  public View getView(int i, View view, ViewGroup viewGroup) {
3      View v = layoutInflater.inflate(R.layout.listentry_spielerverwaltung, null);
4
5          Spieler dieserSpieler = alleSpieler.get(i);
6
7          TextView tvName = v.findViewById(R.id.tvEntryName);
8          tvName.setText(dieserSpieler.getName());
9
10         ...
11
12         return v;
13 }
```

Listing 8.18: Listenelement im Adapter bearbeiten

Neue Spieler können über den Floating-Action-Button hinzugefügt werden. Dieser leitet den Nutzer zu einer Activity weiter, in der dieser den Namen des Spielers eingeben kann.

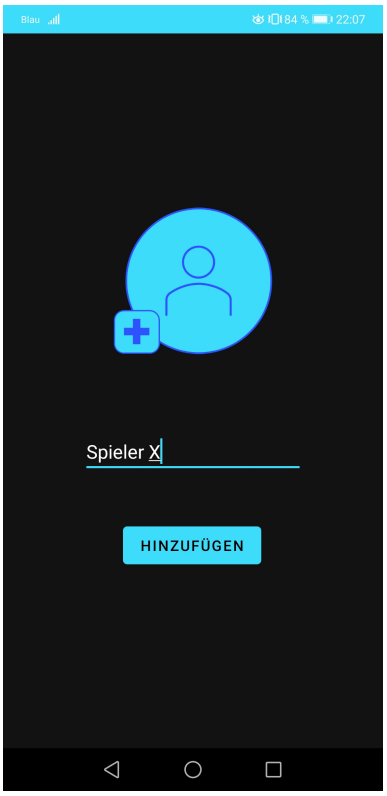


Abbildung 8.16: Spieler hinzufügen

Dabei wird diese Activity mit der Aufforderung aufgerufen ein Ergebnis zu liefern, wie der folgende Auszug zeigt.

```
1  Intent intent = new Intent(getApplicationContext(), AddSpielerActivity.class);
2  startActivityForResult(intent, 0);
```

Listing 8.19: Activity für ein Ergebnis aufrufen

Beim Hinzufügen des Spielers wird im Hintergrund automatisch eine neue Spielernummer erzeugt. Der Name des Spielers wird dann unter dieser Nummer gespeichert. Zudem werden, wie im Quelltext-Auszug 8.20 zu sehen ist, Spielername und Nummer an die Activity der Spielerverwaltung zurückgegeben.

```
String name = etGetName.getText().toString();

if(!name.equals("")) {

      SharedPreferences spSpieler = getSharedPreferences("SpielerNamen", 0);
      int nrSpieler = spSpieler.getInt("AnzahlSpieler", 0);

      SharedPreferences.Editor myEditor = spSpieler.edit();
      myEditor.putString("Spieler_" + (nrSpieler + 1), name);
      myEditor.putInt("AnzahlSpieler", ++nrSpieler);
      myEditor.commit();

      Intent resultIntent = new Intent();
      resultIntent.putExtra("NAME", name);
      resultIntent.putExtra("ID", (nrSpieler + 1));
      setResult(RESULT_OK, resultIntent);

      finish();
} else {
            ...
            finish();
}
```

Listing 8.20: Spielernamen Speichern

Zurück in der Activity der Spielerverwaltung wird die Methode „onActivityResult“ aufgerufen, in der die Ergebnisse verarbeitet werden.

```
rotected void onActivityResult(int requestCode, int resultCode, Intent data) {
      super.onActivityResult(requestCode, resultCode, data);

      if(requestCode == 0 ){

            if(resultCode == RESULT_OK){

                  String name = data.getStringExtra("NAME");
                  int id = data.getIntExtra("ID", -1);

                  if(!(id == -1)){
                        alleSpieler.add( new Spieler(name , id) );
                        SpielerAdapter spielerAdapter =
                              new SpielerAdapter(this, alleSpieler);
                        spielerListe.setAdapter(spielerAdapter);
                  }
            }
      }
}
```

Listing 8.21: Neuen Spieler der Liste hinzufügen

Der Spieler wird der Liste hinzugefügt und diese wird aktualisiert, damit der Nutzer den neuen Spieler gleich in der Liste findet. Da mit dieser App noch keine Messungen erfolgen, und diese App nur den Prototypen darstellt, sind der Spielerverwaltung keine weiteren Funktionen hinzugefügt worden. Diese sind bei der Weiterentwicklung zu implementieren. Dazu muss auch im Zuge dessen eine Speicherverwaltung entwickelt werden. Die Verwendung der „SharedPreferences“ ist nur übergangsweise für diesen Prototypen gedacht, da diese nicht dazu gemacht sind große Datenmengen zu speichern.

# 9 Kritische Betrachtung der Ergebnisse

Abschließend wird die Arbeit an dieser Stelle noch einmal zusammengefasst. In dieser Arbeit wurden Methoden der Bildverarbeitung entwickelt, mit denen es möglich ist geworfene oder geschossene Bälle in einer Bildsequenz zu detektieren und deren Geschwindigkeit zu bestimmen. Zusätzlich wurde der Prototyp einer App entwickelt, in der diese Algorithmen eingefügt werden können.
Insgesamt ist mit dieser Arbeit das Grundgerüst einer mobilen Anwendung entstanden, mit der es möglich ist Ballgeschwindigkeiten zu messen, indem die Bewegung seitlich, aus der Hand aus aufgenommen wird.

Für den Teil der Bildverarbeitung wurden leistungsstarke und effiziente Algorithmen entwickelt. Mit dem allgemeinen Algorithmus können Bälle aller Art detektiert werden. Dazu müssen ein paar Parameter angepasst werden, je nachdem was für Bälle erkannt werden sollen. Bei größeren Bällen und keinen bis leichten Verwacklungen funktioniert die Detektion sehr genau und zuverlässig. Umso kleiner die Bälle sind, umso weniger darf das Bild verwackelt sein, bevor der Ball nicht mehr erkannt wird. Doch auch hier erfolgen die Detektionen zuverlässig, wenn entsprechend vorsichtig gefilmt wird. In Sachen Genauigkeit übertrifft der Algorithmus das Radar-Messgerätes und erlaubt so eine noch präzisere Messung. Auch der optimierte Algorithmus erlaubt eine präzisere Messung als das Radar-Messgerätes. Aufgrund der verwendeten Berechnungen zeigt dieser allerdings Abweichungen gegenüber dem allgemeinen Algorithmus. Diese fallen jedoch gering aus, im Vergleich zu den des Radar-Messgerätes. Die Verfolgung der Ballpositionen erfolgt zuverlässig und ist sehr gut für den geplanten Einsatz geeignet. Nur in manchen Situationen, wenn es beispielsweise zu Anfang des Schusses Störungen im Suchbereich gibt, kann es dazu kommen, dass versehentlich der hintere Ball der Doppelerscheinungen verfolgt wird. Dies ist jedoch selten der Fall und führt zu geringen Abweichungen, die im Vergleich zu den ohnehin auftretenden Abweichungen, vernachlässigbar sind.
Beide Algorithmen schaffen eine solide Grundlage für die Implementierung in einer mobilen Anwendung.

Der Prototyp der App bietet einen guten Einblick darin, wie die fertige App aussehen könnte und auf welche Aspekte zusätzlich zu achten ist. Dieser Prototyp enthält die grundlegenden Funktionen, die eine spätere Version der App bieten soll und vermittelt so ein gutes Gefühl für den Umgang mit dieser App. Zusammen mit den Algorithmen ist so der Grundbaustein für dieses Projekt geschaffen.

Hinzukommt der Transfer von Wissen bezüglich der Kamerainterfaces in Android, der Frage wie die Bildverarbeitung in Android implementiert werden kann und die theoretische Betrachtung der Abweichungen, die bei den Messungen entstehen können. Dieses Wissen ermöglicht den Weiterentwicklern einen schnellen und gezielten Einstieg in das Projekt.

## 9.1 Anforderungen

An dieser Stelle wird noch ein Blick auf die Anforderungen aus Kapitel 4.3 geworfen, um zu betrachten, in welchen Umfang diese umgesetzt werden konnten.

F1 Mit dem allgemeinen Algorithmus können Bälle aller Sportarten detektiert und gemessen werden. Um ein besonderes Augenmerk auf den Fußball zu legen, wurde ein separater Algorithmus entwickelt. Diese Anforderung wurde somit erfüllt.

F2 Der Prototyp wurde in der Android Version 8 programmiert. Alle benötigten Interfaces und Bibliotheken können für Android 8 verwendet werden. Die Anforderung wurde von Seiten dieser Arbeit erfüllt, muss jedoch auch von den Weiterentwicklern bedacht werden.

F3 Da die Algorithmen nicht in der App implementiert wurden, ist diese Anforderung im Zuge dieser Arbeit nicht erfüllt.

F4 Diese Anforderung konnte bereits innerhalb dieser Arbeit realisiert werden, wie in Kapitel 8.6 beschrieben ist.

F5 Bei der Entwicklung der Algorithmen wurde auf die Verwendbarkeit von Bildsequenzen, die aus freier Hand aufgenommen wurden, großen Wert gelegt. Die Algorithmen wurden dabei so entwickelt, dass sie bis zu einem gewissen Maß Verwacklungen zulassen. Somit können sie bei einer ruhigen Handhaltung verwendet werden. Diese Anforderung ist also als erfüllt zu betrachten.

F6 Auch diese Anforderung ist noch nicht erfüllt, da die Bildverarbeitung noch nicht in die App integriert ist. Jedoch verfügt der optimierte Algorithmus bereits über eine Überprüfung, ob die Messung fehlerhaft gewesen sein könnte. Beim allgemeinen Algorithmus steht diese noch aus.

F7 Aus dem gleichen Grund wie bei F6 und F3 wurde diese Anforderung noch nicht erfüllt.

N1 Eine intuitive und einfachen Nutzung ist immer auch Gegenstand der subjektiven Wahrnehmung. Nichtsdestotrotz wurde das Konzept dieser App unter diesem Gesichtspunkt entwickelt. Es wurde ein minimalistisches Design verwendet und keine unnötigen Verzweigungen eingebaut. Die Activities der Bildverarbeitung, die eventuell etwas komplizierter sind, wurden trotz möglichst einfach gehaltenen Design zusätzlich erklärt. Zusammenfassend kann diese Anforderung somit als erfüllt gewertet werden.

N2 Wie in Kapitel 7.4 besprochen, sind die Algorithmen im Begriff der Echtzeit, der im Kontext dieser Arbeit verwendet wird, nicht echtzeitfähig. Bei der Entwicklung wurde auch auf die Rechendauer geachtet und mit der Rechenleistung des Rechners ist diese Anforderung erfüllt. Ob diese Anforderung auch bei der Implementierung auf einem rechenschwächeren Smartphone erfüllt ist, ist noch zu testen. Mit den benötigten Zeiten, die in Kapitel 7.4 beschrieben wurden, kann jedoch eine Abschätzung erfolgen. Wird zum Beispiel eine Bildsequenz mit 150 Bildern aufgenommen, so braucht der Laptop, bei einer Bearbeitungszeit von 0,05 Sekunden pro Bildpaar, etwa 7,45 Sekunden. Um die Anforderungen zu erfüllen, darf das Smartphone also maximal 3,35-mal so lange brauchen, wie der Laptop.

N3 Die App wurde so programmiert, dass neue, relevante Daten sofort abgespeichert werden. Für diesen Prototypen ist damit diese Anforderung erfüllt. Bei der Weiterentwicklung der App muss dieser Punkt weiter mit eingeplant werden.

N4 Dieser Punkt wurde durch ein ansprechendes Design und durch das Verwenden von Grafiken erfüllt.

N5 Bei der Eingrenzung des Suchbereiches bei der optimierten Bildverarbeitung, wurde die in Kapitel 5.1.1 gewonnenen Erkenntnisse genutzt. Theoretisch ist die Berechnung des Suchbereiches damit für Schüsse mit einer Geschwindigkeit von 120 km/h

ausgelegt. Dies konnte aber nicht getestet werden. Im Zweifelsfall gibt es bei der allgemeinen Bildverarbeitung keine theoretische Begrenzung durch den Suchbereich. Diese Anforderung kann unter Vorbehalt als erfüllt betrachtet werden.

N6 Die optimierte Bildverarbeitung wurde auch anhand Videos von Schüssen aus 11 Metern Entfernung entwickelt. Diese konnten auch erfolgreich analysiert werden. Diese Anforderung ist aus Sicht dieser Arbeit somit erfüllt.

N7 Diese Anforderung ist durch die optimierte Bildverarbeitung erfüllt.

N8 In den Tests der Bildverarbeitungen hat sich gezeigt, dass die Bälle so gut detektiert werden, dass es zwischen der Auswertung durch die Algorithmen und durch eine Auswertung per Hand praktisch keine Unterschiede gibt.

Zusätzlich waren noch andere Anforderungen zu bedenken, die nicht Bestandteil der App waren. Eine große Rolle hierbei spielen die Anforderungen der Weiterentwickler, die später dieses Projekt zu Ende führen werden. Eine wichtige Anforderung für diese ist die Weitergabe von Wissen. Um diese Anforderung bestmöglich zu erfüllen, wird den Weiterentwicklern neben diese Arbeit noch weiteres Material zur Verfügung gestellt. Zu diesem Material gehören unter anderem die Python Skripte der Bildverarbeitung, Android Projekte für die einzelnen Kamera-Interfaces und das Projekt des Prototypen.

## 9.2 Ausblick

Abschließend wird es an dieser Stelle ein Überblick über mögliche Weiterentwicklungen und Verbesserungen des Projektes geben. Der zentrale Punkt der Weiterentwicklung wird die Übersetzung der Algorithmen in C++ und die anschließende Integration dieser in die App sein. Im Zuge dessen ist die App um die benötigten Funktionen weiterzuentwickeln. Etwa muss die Spielerverwaltung um die in den Anforderungen beschriebenen Funktionen erweitert werden. Dazu ist auch eine Speicherverwaltung zu entwerfen und zu implementieren. Aber auch bei den Algorithmen bieten sich weitere Möglichkeiten zur Verbesserung. Beispielsweise ist die im Konzept besprochene Möglichkeit einer softwaretechnischen Bildstabilisierung beim allgemeinen Algorithmus aus zeitlichen Gründen nicht implementiert worden. Dies könnte die Robustheit gegenüber Verwacklungen weiter erhöhen und wäre auch für den optimierten Algorithmus eine denkbare Möglichkeit Trefferquote und Genauigkeit weiter zu optimieren. Ein weiterer Aspekt, der untersucht werden könnte, ist, ob sich aus den Abbildungen des Balles auf den Differenzbildern nicht

doch auf die seitliche Bewegung des Balles geschlossen werden kann. So könnte auch die in Kapitel 5.2 besprochenen Abweichungen reduziert werden. Um den Weiterentwicklern einen möglichst leichten Einstieg in die Arbeit bieten zu können, sind diverse Dateien dieser Arbeit beigefügt. Im Anhang dieser Arbeit ist eine ausführliche Beschreibung dieser.

# Literaturverzeichnis

[1] Griffiths, Dawn ; Griffiths, David: *Head First Android Development A Brain-Friendly Guide*. O'Reilly, 2017. – 2. Auflage. – ISBN 978-1-491-97405-6

[2] Joshi, Prateek: *OpenCV with Python By Example*. Packt Publishing, 2015. – ISBN 978-1-78528-393-2

[3] Mark James Burge, Wilhelm B. und: *Digitale Bildverarbeitung – eine algorithmische Einführung mit Java*. Springer Vieweg, 2015. – ISBN 978-3-642-04603-2

# A Anhang

## A.1 Über die beigefügten Dateien

Dieser Abschnitt gibt einige Informationen zu den Dateien, die mit dieser Bachelorarbeit zusammen abgegeben wurden. Neben den Endversionen der entwickelten Quelltexte, handelt es sich um weitere Dateien, die den Weiterentwicklern die Einarbeitung erleichtern sollen.

### A.1.1 Android Projekte

Insgesamt vier Android Projekte sind im Zusammenhang mit dieser Arbeit abgegeben. Drei dieser vier Projekte enthalten jeweils nur eine Activity mit einer Kameravorschau.

**Camera2** Enthält den gesamten Quelltext, der für die Implementierung der Camera2 API notwendig ist.

**CameraX** Enthält den gesamten Quelltext, der für die Implementierung der CameraX API notwendig ist.

**OpenCV** Enthält den gesamten Quelltext, der für die Implementierung des OpenCV Kamerainterfaces notwendig ist.

Bei dem letzten Projekt handelt es sich um:

**Prototyp** Ist der Prototyp der App.

### A.1.2 Python Skripts

Folgende Python Skripts sind dieser Arbeit beigefügt:

**allgemein** Quelltext der allgemeinen Bildverarbeitung. Zum Ausführen dieses Skripts ist eine bestimmte Ordnerstruktur nötig. Diese wird aus den Konstanten im ersten Quelltextabschnitt ersichtlich.

**clearVolume** Speichert eine tonlose Kopie eines Videos.

**erstesBildSpeichern** Speichert das erste Bild aus einem Video als jpg-Datei ab. Dieses Skript wurde genutzt, um die Bilder zu erzeugen, aus denen die Werte ausgelesen wurden, die für den optimierten Algorithmus benötigt werden.

**optimiert** Quelltext der optimierten Bildverarbeitung. Zum Ausführen dieses Skripts ist eine bestimmte Ordnerstruktur nötig. Diese wird aus den Konstanten im ersten Quelltextabschnitt ersichtlich.

**trackingAlgorithmen** Dieses Skript enthält alle drei, in Kapitel 7.3.5 beschriebenen Tracking-Algorithmen. Dieses Skript wurde genutzt, um die Tracking-Algorithmen zu testen und zu vergleichen. Daraus entstand unter anderem Abbildung 7.20.

**TrackingGrid** Dieses Skript wurde verwendet, um mit dem dritten Tracking-Algorithmus (vgl. Kapitel 7.3.5) die Daten für die Abbildung 7.18 zu erzeugen.

**TrackingZeitTest1** und **TrackingZeitTest2** Diese Skripts wurden verwendet, um die in Abbildung 7.21 zu sehende Grafik zu erzeugen. Das Skript TrackingZeitTest1 hat die dazu benötigten Daten erzeugt und TrackingZeitTest2 aus diesen Daten die Grafik.

### A.1.3 Matlab Skripts

Folgende Matlab Skripts sind dieser Arbeit beigefügt:

**Abweichungen** Enthält sämtliche in Kapitel 5 beschriebenen Berechnungen zu den Abweichungen, die während des Schusses entstehen können. Alle in diesem Kapitel verwendeten Grafiken wurden mit diesem Skript erzeugt.

**AuswertungTrackingZeit** Mit diesem Skript wurden die Daten aus dem Python Skript trackingAlgorithmen ausgewertet, sodass die Abbildung 7.20 erzeugt werden konnte.

**TrackingGrid** Erzeugt die in Abbildung 7.18 zu sehenden Mesh-Plots aus den Daten des Python Skripts TrackingGrid.

**TrackingGridInvertiert** Erzeugt einen invertierten Plot der durch TrackingGrid erzeugten Mesh-Plots. Bei diesen stellt die Koordinate mit der geringsten Abweichung das Maximum des Mesh-Plots dar.

### A.1.4 Videos

Um den Weiterentwicklern den Einstieg zu erleichtern, wurden auch einige Videos mit abgegeben. Diese unterteilen sich in 2 Kategorien:

**Schüsse** Abgegeben sind 23 aufgenommene Schüsse inklusive der Textdatein, die die nötigen Werte enthalten. Enthalten sind unter anderem die fünf Videos, die in Kapitel 7 zum abschließenden Testen verwendet wurden.

**Tracking** Enthält zwei der Videos, die für die Tests der Tracking-Algorithmen genutzt wurden. Enthalten sind auch die entsprechenden Textdateien.

### A.1.5 Weitere Dateien

Zusätzlich sind weitere Dateien mit beigelegt, die für die Weiterentwickler nützlich sein könnten.

**BilderPrototyp** Microsoft Word Dokument mit den Grafiken, die für die Prototyp-App verwendet wurden.

**OpenCV2Android** Ausführliche Anleitung mit Screenshots, wie die OpenCV Bibliothek zum Android Projekt hinzuzufügen ist, damit das Kamerainterface genutzt werden kann, wie etwa in OpenCV und Prototyp.

**AblaufOptimiert** Grafik des Ablaufes des optimierten Algorithmus. Da die Struktur dieses Algorithmus sehr komplex ist, bietet diese Grafik ein schnelles Verständnis des Anlaufes. Die orange dargestellten Verbindungen sind dabei nur für die Implementierung auf dem Computer gedacht, um alle Bilder des ursprünglichen Videos in das ausgewertete zu speichern. In der Implementierung in der App können diese entfallen.

## Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

| Ort | Datum | Unterschrift im Original |
|---|---|---|