Studienarbeit Dokumentation

Video-Analyse: Badmintonball-Tracking

Johanna Sommer

Fakultät Informatik

DHBW Stuttgart

Betreuer: [redacted]

Juni 2019

Selbstständigkeitserklärung

Ich versichere hiermit, dass ich die Studienarbeit mit dem Thema:

Video-Analyse: Badmintonball-Tracking

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.
Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Zürich, 2. Juni 2019 _____

Ort, Datum Unterschrift

Anhang

Der Code des Projekts wurde in Form einer .zip-Datei bei der Abgabe bereitgestellt, alle für diese theoretische Bearbeitung relevanten Code-Ausschnitte sind jedoch in den entsprechenden Kapiteln dargestellt.

Weiterhin wurde mit dieser Arbeit eine .zip-Datei abgegeben, die alle Original-Videos, Ausschnitte und Dateien enthält.
Dieses Archiv ist wie folgt aufgebaut:

```
archive
├── final visualization
├── process
│   ├── backsub
│   ├── data
│   ├── full
│   └── snips
└── test frames
    ├── blobdec test 1
    └── blobdec test 2
```

Im Ordner `final visualization` befinden sich alle in 3D visualisierten Ballwechsel. Der Ordner `process` enthält in `full` die kompletten Aufnahmen beider Kameras, in `snips` die Aufnahmen zugeschnitten in Ballwechsel, in `backsub` die Aufnahmen nach der Background Subtraction und in `data` die relevanten CSV-Dateien. Im Ordner `test frames` befinden sich die Testbilder, die zur Parameterauswertung benutzt wurden.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

1.1 Kontext

Technologie ist im Laufe der letzten Jahr zum festen Bestandteil jeden Aspekts unseres alltäglichen Lebens geworden. Es gibt unzählige Beispiele, wie neue Entwicklungen der IT uns lästige Aufgaben abnehmen und neue Möglichkeiten eröffnen. Selbst bei Sport, einem Gebiet, das auf menschlichem Können, Training und strategischem Denken basiert, halten neue Technologien ihren Einzug.

Die am meisten in professionellem Sport benutze und die wohl meist bekannte Technologie ist das sogenannte Hawk-Eye System von Hawkeye Technologies. Seit der Saison 2015/2016 ist es in der Bundesliga im Einsatz und wird dort für strittige Torentscheidungen zurate gezogen.

In mittlerweile drei von vier Grand Slam Turnieren wird diese Technologie eingesetzt, um knappe Entscheidungen abzusichern - so zum Beispiel ob ein Ball im 'Aus' ist oder nicht. Seit Ende 2013 wird auch für Linien-Entscheidungen im Badminton das System von Hawkeye Technologies verwendet.
Bei Sportübertragungen von Sky Sports und BBC findet es ebenfalls Anwendung, dort für Animation der vergangenen Spielzüge und ergänzende statistische Informationen [1].

Ein Augenmerk auf die statistische Analyse legt seit 2018 auch IBM und stellt für die Grand Slam Turniere den IBM SlamTracker bereit. Das Tool greift auf 12 Jahre Turnierdaten zurück und bietet Real-Time Statistiken für alle Live Spiele. Besonders interessant ist dabei z.B. das 'Momentum-Feature', das zeigt, welcher Spieler momentan einen Vorteil besitzt, basierend auf vorangegangenen Begegnungen und individuellen Stärken [2].

Bis heute ist die Analyse menschlicher Bewegungen im Sport nur durch Videoanalyse und magnetischer Nachverfolgung möglich. Dies ist jedoch meist nur in genau preparierten Turniersituationen möglich und nicht für Amateursportler zugänglich. Nach Fortschritten in Micro-Elektro-Mechanischen-Systemen, auch MEMS Technologien genannt, eröffnen sich nun auch immer mehr Möglichkeiten in der Benutzung von am Körper befestigten Sensoren für die Bewegungsanalyse im Sport [3].

Mit Aufnahmemöglichkeiten bald auch zugänglich für Amateursportler und den Möglichkeiten von KI für die Analyse spielt Technologie auch für traditionelle Sportarten eine immer wichtigere Rolle.

1.2 Aufgabenstellung

Ziel dieser Studienarbeit ist es, Videomaterial eines Badmintonspiels auszuwerten, sodass die Flugbahn des Federballs im Raum erkannt und später digital dargestellt wird.

Dazu bedarf es eines Programms, das den sich bewegenden Ball aus dem Bild in irgendeiner Weise extrahiert und dessen Lokation über die Zeit des Ballwechsels festhält. Basierend auf diesen Daten soll dann der Ballwechsel virtuell dargestellt werden, z.B. in Form einer Animation oder eines Videos. Zuletzt ist es ebenfalls Teil der Aufgabe, die Geschwindigkeit einzelner Schläge zu bestimmen.

Es ist für diese Aufgabe nicht erforderlich, dass genannter Prozess in Echtzeit abläuft.

1.3 Voraussetzungen

1.3.1 Badminton

Die Sportart Badminton gehört zu den Ballsportarten. Jeder Spieler hat dabei einen Schläger, mit dem der Federball über das Netz geschlagen werden kann [4]. Im professionellen Badminton gibt es drei Spielarten:

Herren/Damen Einzel: insgesamt zwei Spieler spielen gegeneinander auf dem Einzelfeld

Herren/Damen Doppel: zwei Teams mit jeweils zwei Spielern spielen gegeneinander auf dem Doppelfeld

Mixed Doppel: zwei Teams, bestehend jeweils aus einer weiblichen und einer männlichen Person, spielen gegeneinander auf dem Doppelfeld

Im weiteren Verlauf der Projektarbeit soll ausschließlich die Spielart 'Einzel' betrachtet werden.

Die Abmessungen sowohl des Einzel- als auch des Doppelfelds sind, wie auf Abbildung 1.1 zu sehen, genau von dem Deutschen Badminton Verband (DBV) definiert [4]. So ist das Feld ein Rechteck, das durch 40 mm breite Linien begrenzt ist. Das Einzelfeld ist 5.16 m breit und 13.4 m lang. Das Netz hängt auf einer Höhe von 1.55 m.

Ein Spiel besteht aus mindestens zwei, maximal drei Sätzen, es gilt das Spielprinzip 'Best of Three'. Ein Satz wird von dem Spieler gewonnen, der zuerst 21 Punkte erreicht. Ein Punkt ist dann erzielt, wenn der Gegenspieler nicht mehr in der Lage ist, den Ball über das Netz zurück in das gegenüberliegende Spielfeld zu schlagen.

Mögliche Szenarien, um einen Punkt zu erzielen sind so z.B.:

- der Gegenspieler erreicht nicht rechtzeitig den Ball und er fällt in seiner Spielfeldhälfte auf den Boden
- der Gegenspieler spielt den Ball zurück über das Netz, dieser fällt aber außerhalb des Spielfelds auf den Boden
- der Gegenspieler schlägt den Ball ins Netz

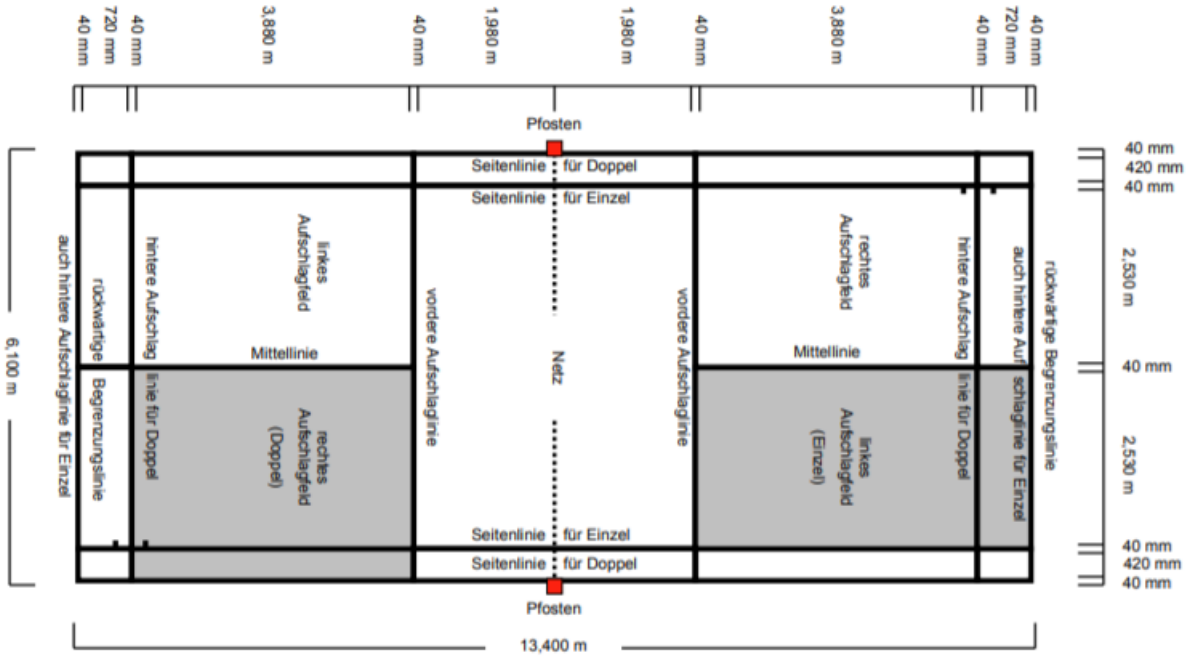

Abbildung 1.1: Badmintonfeld nach DBV [4].

Der Ball berührt während eines Ballwechsels nicht den Boden. Passiert dies, ist der Ballwechsel vorbei und je nach Ort des Aufpralls wird der Punkt vergeben. Danach beginnt ein neuer Ballwechsel; der Spieler, der den letzten Punkt erzielt hat, macht dabei den Aufschlag.

Beachtenswert ist bei Badminton die vergleichsweise kleine Größe des Balls. Dieser besteht aus einem Kork-Kopf und 16 Federn. So wiegt er circa 5 Gramm und ist zwischen 62 mm und 70 mm lang [4]. Federbälle werden mit einer Geschwindigkeit von bis zu 490 km/h geschlagen, was Badminton zum schnellsten Rückschlagspiel der Welt macht [5].

1.3.2 Hardware

Für dieses Projekt wurden zwei sogenannte 'Action-Cams' des gleichen Typs verwendet, deren Auflösung 4K - auch Ultra HD genannt - beträgt. Dies bedeutet 4.096 x 2.160 Pixel pro einzelnem Bild [1].

[1] Für die Aufnahmen wurde die Kamera nicht in den 4K-Modus geschaltet, weswegen die Aufnahmen nur eine Auflösung von 1.920 x 1.080 Pixeln haben

Videos werden von diesen Kameras mit einer Frame-Rate von 60 Bildern pro Sekunde und im Winkel von 170 aufgenommen. Für diesen Kameratyp werden zur Speicherung SD-Karten mit einer Schreibleistung von 80-100 MB/s benötigt, konkret werden hier zwei 32GB SD-Karten von SanDisk mit einer Schreibleistung von 95 MB/s benutzt.

Als die Aufnahmen gemacht wurden, kam es allerdings zu Problemen, die nun beschrieben werden sollen. Diese Probleme werden hier erwähnt, da sie die Vorgehensweise des Projekts sowie die Resultate massiv beeinflussen. Es war im vorgegebenen Rahmen des Projekts nicht möglich, die Aufnahmen erneut anzufertigen und den Fehler zu beheben. Während die eine Kamera fehlerfreie Aufnahmen lieferte, konnte die andere Kamera nicht die Aufnahmequalität bereit stellen, die für die weitere Vorgehensweise benötigt wurde. Bei diesen Aufnahmen kam es zu sogenannten Bildrauschen (Englisch: Noise), einer Störung bei der das Bild zu flimmern scheint. Dies kann zum Beispiel durch unzureichende Belichtung auftreten.

Um diesem Bildrauschen teilweise entgegenzuwirken, wurden die Videos mit Adobe AfterEffects bearbeitet. Dort wurde der Effekt 'Video rauschen entfernen' in hoher Intensität angewendet und danach der Kontrast erhöht. Weiterhin wurde der Stabilisierungseffekt angewandt, sodass leichte Erschütterungen der Kamera keine Pixelveränderungen auslöst. Es wurden alle Videos anschließend in MP4 exportiert.

1.3.3 Positionierung

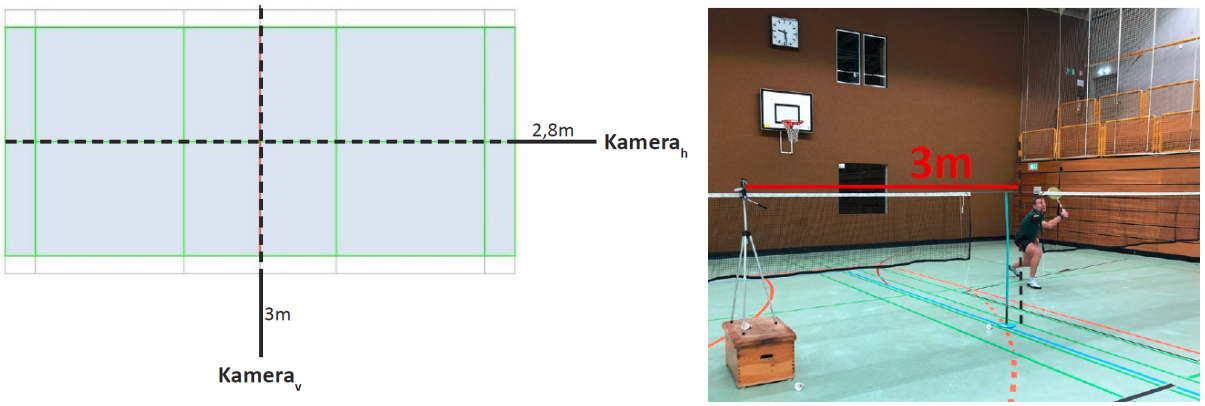

Abbildung 1.2: Positionierung der Kameras

Die Kameras werden für die Aufnahmen orthogonal zueinander am Feldrand aufgestellt, wie Abbildung 1.2 zeigt. Die horizontale Kamera, im weiteren $Kamera_h$ genannt, ist dazu an der Netzlinie ausgerichtet, 3 Meter von der Doppel-Seitenlinie entfernt. Die vertikal positionierte Kamera, im weiteren $Kamera_v$ genannt, ist an der Mittellinie ausgerichtet, 2,8 Meter entfernt von der rückwärtigen Begrenzungslinie. Beide Kameras befinden sich auf einer Höhe von 1,45 Metern.

Aus dieser Weise der Positionierung ergibt sich Videomaterial für alle drei Dimensionen des Spielfelds. $Kamera_h$ nimmt die Breite des Spielfelds auf, sowie die Höhe, während $Kamera_v$ eine Perspektive auf die Längs-Dimension sowie ebenfalls auf die Höhe gibt.

Kapitel 2

Wissenschaftlicher Teil

2.1 Background Subtraction

2.1.1 Theorie

Background Subtraction ist ein wichtiger Vorverarbeitungs-Schritt im Bereich der Video-Analyse mit dem Ziel, den sich bewegenden Vordergrund eines Videos von dem statischen Hintergrund zu unterscheiden, zum Beispiel ein Auto, das über eine Autobahn fährt [6] [7]. Allgemein gesagt wird bei Background Subtraction eine Subtraktion des aktuellen Frames von einem Modell des Hintergrunds durchgeführt, übrig bleiben die sich bewegenden Objekte in dem Video [8].

Die Grundlage dafür ist das sogenannte Gaussian Mixture Model [6]. Die Aufgabe, ein Model für den Hintergrund zu berechnen, ist in seiner einfachsten Form beschrieben durch die Formel

$B(x, y, t) = \frac{1}{t} \sum_{t'=1}^{t} I(x, y, t)$

in welcher $I(x, y, t)$ den momentanen Wert des Pixels bei (x, y) zum Zeitpunkt t beschreibt [6]. So sagt diese Formel also lediglich aus, dass das Hintergrundmodell der Durchschnitt des Bilds über eine bestimmte Zeit ist.

Diese einfache Annahme hält aber nicht stand, wenn sich in dem Video zum Beispiel über die Zeit die Lichtverhältnisse ändern. Deshalb wird die Formel mit der Gewichtung α erweitert, welche für ein Bild exponentiell kleiner wird, je mehr das einzelne Bild in die Vergangenheit rückt. Dies wird von Friedman & Russel auch als 'exponentielles Vergessen' beschrieben [6]:

$B(x, y, t) = (1 - \alpha) * B(x, y, t - 1) + \alpha * I(x, y, t).$

Da das Modell des Hintergrundes stetig aktualisiert werden muss, ist dieser Algorithmus den Online-Learning Algorithmen zuzuordnen.

Nun gilt es zu entscheiden, ob ein Pixel dem Hintergrund angehört oder ein bewegtes Objekt ist. Dazu eignet sich wieder das Beispiel der Autobahn: ist ein einzelnes Pixel an der selben Stelle in seinem 'normalen' Hintergrundzustand, wird es vom Schatten eines Autos beeinflusst und gehört trotzdem noch zum Hintergrund oder gehört es zu einem Auto [6]?

Um diese Frage zu beantworten charakterisiert das Gaussian Mixture Model jedes Pixel anhand zwei Faktoren. Der erste Faktor ist das Intensitätslevel des Pixels, der zweite sind seine RGB-Werte. Mit diesen Werten wird für jedes Pixel eine Verteilung (diese kann normal, gaussch' o.ä. definiert sein) über die Zeit produziert, anhand welcher schlussendlich entschieden wird, ob das Pixel der Vordergrundmaske oder dem Hintergrund zuzuordnen ist [6]. Es gibt einige Varianten dieses Algorithmus, das Framework OpenCV hat diese so implementiert, dass auf sie mit einem einfachen Interface zugegriffen werden kann [7]. Im Weiteren sollen diese Ausprägungen betrachtet werden und schlussendlich die Entscheidung für einen dieser Algorithmen begründet werden.

2.1.2 BackgroundSubtractorMOG and MOG2

Der in OpenCV implementierte 'BackgroundSubtractorMOG' basiert auf einem Gaussian-Mixture Background Segmentation Algorithmus, beschrieben von KadewTraKuPong und Bowden [9]. Dort wird für jedes Pixel nicht nur eine Verteilung angelegt, sondern eine Kombination aus K Gausschen Verteilungen. Die Gewichtungen innerhalb dieser Kombination repräsentieren dabei die zeitlichen Proportionen mit denen die Farben in dem Video existieren. In OpenCV können Paramter, wie die Anzahl K der Verteilungen oder die Länge der Modell-Historie beliebig eingestellt werden.

Der Background Subtractor MOG2 ist eine verbesserte Version des oben vorgestellen MOG, basierend auf der Arbeit von Z. Zivkovic [10]. Im Unterschied zu seinem Vorgänger wird bei MOG2 die Anzahl der Gausschen Verteilungen K automatisch und für jedes Pixel individuell bestimmt. Daraus ergibt sich der große Vorteil, dass sich der Algorithmus besser an sich ändernde Szenen und Hintergründe, durch zum Beispiel Änderungen der Lichtverhältnisse, anpassen kann.

2.1.3 BackgroundSubtractorGMG

Dieser Algorithmus ist eine Kombination aus statistischer Hintergrundmodell-Schätzung und Bayes'scher Pixel Segmentierung [11]. Dafür werden die ersten paar (default: 120) Frames des Videos für die Modellierung des Hintergrunds benutzt. Die Bestimmung des Vordergrunds erfolgt nicht binär, sondern es wird eine Wahrscheinlichkeit angegeben, mit der ein Pixel zum Vordergrund gehört. Dies wird durch Bayes'sche Inferenz umgesetzt. Auch dieser Algorithmus versucht mit sich ändernden Szenen durch z.B. Änderung der Lichtverhältnisse umzugehen, indem er die Gewichte, erwähnt in dem Gaussian Mixture Modell, besonders priorisiert. Neue Hintergrundbilder werden stärker priorisiert als in anderen vergleichbaren Algorithmen [11].

Besonders bei diesem Algorithmus ist, dass Filter-Operationen angewendet werden, um Störungen oder 'Noise' zu entfernen. Ein großer Nachteil des BackgroundSubtractorGMG ist jedoch, dass, da die ersten Frames zur Hintergrundmodellierung benutzt werden, ein schwarzer Bildschirm während der ersten paar Frames erscheint.

2.1.4 Auswahl

In dem Kontext, einen Ballwechsel zu analysieren, ist jeder Frame von essenzieller Bedeutung. Manche Ballwechsel sind innerhalb von weniger als einer Sekunde vorbei und es ist deshalb nicht möglich, einige dieser Frames für die Berechnung des Hintergrundmodells zu opfern. Deswegen ist der Background Subtractor GMG ungeeignet.

Wegen den oben beschriebenen Problemen mit einer der Kameras ist ein wichtiges Kriterium, wie gut der Algorithmus das Störrauschen der Originalaufnahmen ausgleichen kann. Das Rauschen drückt sich in Änderungen der Pixelwerte aus, weswegen bei der Background Subtraction viele einzelne Pixel als bewegt erkannt werden, obwohl in der Realität an diesem Punkt keine Bewegung stattfindet.
Aus diesem Grund wurde schlussendlich Background Subtractor MOG2 ausgewählt, da dort nicht nur sich ändernde Lichtverhältnisse ausgeglichen werden, sondern durch die gleiche Methode auch dem Flimmern zumindest in Teilen entgegengewirkt wird.

Schlussendlich muss noch erwähnt werden, dass Background Subtraction pro Frame abläuft, das heißt, es wird ein Frame eingelesen, dann die Vordergrundmaske und der Hintergrund berechnet, dieser Frame wird in einem neuen Video angehängt und abgespeichert, und so weiter. Da diese Berechnung zeitaufwändig ist, ist das Output-Video der Background Subtraction um den Faktor 0.5 langsamer als das Orgiginalvideo. Das Output-Video hat eine Framerate von 30 Frames pro Sekunde, es geht aber kein Frame verloren, es wird lediglich das ursprüngliche Video verlangsamt.

2.2 Blob Detection

2.2.1 Theorie

Da nun die sich bewegenden Objekte extrahiert werden können, gilt es, den Badmintonball von zum Beispiel den Spielern zu unterscheiden. Hier kann Blob Detection verwendet werden. Ein Blob ist beschrieben als eine Gruppierung verbundener Pixel, die eine Gemeinsamkeit haben, z.B. den Graustufenwert [12]. Um diese zu erkennen durchläuft der Algorithmus 4 Schritte:

Thresholding:
Das originale Bild, in diesem Fall ein Frame, wird inkrementell zu mehreren binären Bildern konvertiert

Grouping:
In jedem binären Bild werden dann verbundene Pixel zu binären Blobs gruppiert

Merging:
Die Mitten der binären Blobs werden ermittelt und Gruppierungen die näher als ein bestimmter Wert zusammen sind werden zusammengefasst

Center & Radius Calculation:
Die Mitten und Radii der neuen Blobs werden ermittelt [12]

Dieser Algorithmus ist auch in OpenCV vorhanden. Dort werden die Blobs in Form ihrer Lokation und ihrem Radius zurückgegeben.

2.2.2 Parameter

Weiterhin können Parameter gesetzt werden, um die Blob-Erkennung zu spezifizieren. Dazu gibt es die Parameter Area, Circularity, Convexity und Inertia [12]. Im Weiteren wird die Bedeutung dieser Parameter beschrieben und an Testreihen gezeigt, wie die Werte dieser Parameter ermittelt wurden.

Hier ist anzumerken, dass eine komplette Auswertung möglicher Werte in dem Zeitrahmen dieses Projekts nicht möglich war. Für jeden Parameter wird jeweils ein Maximum- und ein Minimum-Wert gesetzt und für jeden Wert ergeben sich mindestens 5 Abstufungen. Dies ergibt eine 8x5 Matrix an Konfigurationen, die mit mehreren Bildern ausgewertet werden müsste, was nicht möglich war. Stattdessen wurden die Parameter im Kontext der Anwendung ausgewählt und nur feine Unterschiede wurden durch Tests evaluiert und verifiziert. Die Tests wurden mit dem Code des Software-Entwurfs durchgeführt und von Hand ausgewertet. Die Test-Frames sind in dem Archiv der Videoaufnahmen beigefügt.

Zu beachten ist, dass die Kameras zwei unterschiedliche Perspektiven auf das Feld zeigen und sich somit auch die Parameter je nach Perspektive ändern. In den Tests wurde das ebenfalls berücksichtigt, jeweils die ersten 9 Frames gehören zu Aufnahmen von $Kamera_h$, die nächsten 9 Frames gehören zu Aufnahmen von $Kamera_v$.

Area

Der Parameter $Area$ beschreibt die Größe eines Blobs in Pixeln [12].

Test-Frame	min. 70		70 - 700		100-900	
	Ball erkannt	# andere Elem.	Ball erkannt	# andere Elem.	Ball erkannt	# andere Elem.
1	+	4	+	3	+	2
2	+	4	+	3	+	3
3	+	2	+	1	+	6
4	+	2	+	2	-	1
5	+	5	+	2	-	0
6	+	1	+	1	+	2
7	+	>10	+	>10	+	4
8	+	4	+	4	+	2
9	-	2	-	2	-	4
1	+	>20	+	4	+	10
2	+	>30	-	8	+	5
3	+	>10	-	>10	+	7
4	-	5	-	2	+	2
5	+	6	+	3	-	3
6	+	9	-	4	+	0
7	-	5	+	>10	+	2
8	+	>10	-	0	+	4
9	-	>30	-	4	+	6

Abbildung 2.1: Ergebnis Testreihe Area

Vorallem bei diesem Parameter variieren die gesetzten Minimum- und Maximum-Werte je nach Perspektive sehr. Zu erklären ist dies mit der zurückgelegten Strecke.

Während sich der Ball in Aufnahmen der $Kamera_h$ maximal 5,6 Meter seitwärts bewegen kann (Breite des Felds), sind dies bei $Kamera_v$ 13,4 Meter (Länge des Felds). So kommt es oft vor, dass bei Aufnahmen der $Kamera_v$ der Ball in den einzelnen Frames sehr lang gezogen ist, was sich schlussendlich auf die Erkennung und die Parameterwerte auswirkt.

Abbildung 2.1 zeigt das Ergebnis der Testreihe für den Parameter $Area$. Die höchste Priorität bei der Auswahl der Parameter hat die richtige Ballerkennung. Erst mit zweiter Priorität wird dann die Menge anderer erkannter Elemente (nicht der Ball) berücksichtigt, die möglichst klein sein sollte.

Anhand der farblichen Unterlegung ist schnell zu erkennen, dass für $Kamera_h$ die beste Parameter-Konfiguration für den Parameter $Area$ ein Intervall zwischen 70 und 700 Pixeln ist. Zwar ist die Anzahl an Frames, in denen der Ball erkannt wurde für die ersten beiden Konfigurationen gleich, jedoch wurden in dem Fall, in dem nur das Minimum 70 gesetzt wurde, mehr Elemente fälschlicherweise als Ball identifiziert. Auch für die Konfiguration für $Kamera_v$ ist gut zu erkennen, wieso schlussendlich eine Range zwischen 100 und 900 für den Parameter $Area$ ausgewählt wurde.

Circularity

Der Parameter $Circularity$ beschreibt, wie nah der Umriss eines Blobs zu einem perfekten Kreis ist [12]. Die Kreisförmigkeit ist hier definiert durch

$\frac{4\pi Area}{perimeter^2}$.

Daraus ergibt sich der Wert 1 für einen Kreis, 0.785 für ein Quadrat, usw. [12].

Test-Frame	min. 0.2		min 0.3	
	Ball erkannt	# andere Elem.	Ball erkannt	# andere Elem.
1	+	5	+	3
2	+	6	+	3
3	+	>10	+	1
4	+	2	+	2
5	+	8	+	2
6	+	3	+	1
7	+	>50	+	>10
8	+	9	+	4
9	-	8	-	2
1	+	9	+	4
2	+	5	-	0
3	+	7	-	3
4	+	2	-	2
5	-	3	-	1
6	+	0	+	0
7	+	2	-	1
8	+	4	-	3
9	+	6	+	4

Abbildung 2.2: Ergebnis Testreihe Circularity

Abbildung 2.2 zeigt das Ergebnis der Testreihe für den Parameter $Circularity$. Für Aufnahmen der

$Kamera_h$ zeigt sich, dass die beste Konfiguration lediglich ein Minimumwert von 0.3 ist. Für $Kamera_v$ ist der optimale Wert geringer, wieso, zeigt sehr gut Bild 2.2 aus der Testsuite. Hier ist gut zu sehen, dass die Bälle aus dieser Perspektive nicht immer rund, sondern eher langgezogen erscheinen. Um diese Beobachtung in die Parametereinstellungen miteinzubeziehen, wird hier der Minimumwert auf 0.2 gesetzt. Für beide Perspektiven gibt es keinen Maximumwert, da der Ball durchaus auch optimal rund auf den Aufnahmen erscheinen kann.

Inertia

Der Parameter $Inertia$ beschreibt, wie langgezogen der Blob ist [12]. Die Intertia-Ratio für einen Kreis wäre also 1, für eine Ellipse zwischen 0 und 1 und für eine Linie 0 [12].

Test-Frame	min 0.05		min 0.2	
	Ball erkannt	# andere Elem.	Ball erkannt	# andere Elem.
1	+	3	-	1
2	+	3	+	3
3	+	1	+	1
4	+	2	+	1
5	+	2	+	2
6	+	1	+	0
7	+	>10	+	2
8	+	4	-	4
9	-	2	-	0
1	+	9	+	6
2	+	5	-	2
3	+	7	-	3
4	+	2	-	1
5	-	3	-	1
6	+	0	+	2
7	+	2	+	2
8	+	4	+	6
9	+	6	-	5

Abbildung 2.3: Ergebnis Testreihe Inertia

Die Langgezogenheit der vertikalen Aufnahmen spiegelt sich auch in diesem Parameter wieder, hier wird lediglich ein Minimum von 0.05 gesetzt. Dies lässt langgezogene Bälle zu, exkludiert aber Linien, die z.B. fälschlicherweise als bewegtes Objekt aus der Backround Subtraction kommen. Für die horizontalen Aufnahmen kann der Minimumwert auf 0.2 erhöht werden.

Convexity

Am besten deutlich wird die Bedeutung des Parameters $Convexity$ in Abbildung 2.4. Dort ist zu sehen, dass Konvexität beschreibt, wie ähnlich die engste Hülle um ein Objekt der tatsächlichen Größe ist. Mathematisch definiert ist die Konvexität durch

$\frac{Area_{Objekt}}{Area_{konvexeHuelle}}$ [12].

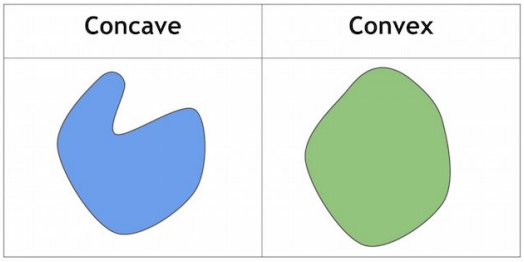

Abbildung 2.4: Erklärung Convexity

Die Tests mit diesem Parameter haben keine positive Auswirkung auf die schlussendliche Analyse gehabt, weshalb dieses Parameter nicht gesetzt wird. Der Grad der Konvexität hat im Kontext dieser Anwendung keine Auswirkung auf die Entscheidung, ob ein Blob extrahiert wird oder nicht.

2.2.3 Ergebnis

Nach der theoretischen Bearbeitung und der Auswertung der optimalen Parameter ergeben sich eindeutige Werte für die Parameter $Area$, $Circularity$ und $Inertia$. Diese werden wie folgt in den Software-Entwurf integriert

```
params = cv.SimpleBlobDetector_Params()
params.filterByArea = True
params.filterByCircularity = True
params.filterByConvexity = False
params.filterByInertia = True
if self.long == True:
    params.minArea = 70
    params.maxArea = 700
    params.minCircularity = 0.3
    params.minInertiaRatio = 0.2
else:
    params.minArea = 100
    params.maxArea = 900
    params.minCircularity = 0.2
    params.minInertiaRatio = 0.05
```

2.2.4 Farbe

OpenCV bietet für Blob Detection einen Parameter Farbe an, der aber inzwischen als veraltet markiert ist und nicht mehr funktioniert. Inspiriert davon wurde aber versucht, diese Funktionalität selbst zu implementieren, da sich der Ball in seiner weißen Farbe doch stark von den restlichen bewegten Objekten unterscheidet.

Dafür werden zwar nur die Blobs berücksichtigt, die bei der Background Subtraction als bewegte Objekte erkannt werden, von diesen werden aber die Pixel bzw. Farben des Originalbilds benutzt. Ein Pixel setzt sich aus drei Werten, dessen R-G-B-Werten, zusammen. Die islight-Funktion, die True zurückgeben soll, wenn der Blob ein Ball ist, besitzt zwei Fälle, in denen True zurückgegeben wird.

```
def islight(x, frame):
    point1 = x.pt
    x1 = int(point1[0])
    y1 = int(point1[1])
    R = int(frame[y1, x1][1])
    B = int(frame[y1, x1][0])
    G = int(frame[y1, x1][2])
    if (R+G+B)/3 > 230:
        print('over: ' + str(R) + " " + str(G) + " " + str(B))
        return True
    if abs(R-B)<50 and abs(R-G)<50 and abs(G-B)<50 and R>120 and G>120 and B>120:
        print(R, G, B)
        return True
    return False
```

Der erste Fall überprüft lediglich, ob die Farbwerte des Pixels denen einer hellen Farbe entsprechen. Der zweite Fall überprüft die Differenz der einzelnen Farbwerte. Damit soll erkannt werden, ob die Farbe des Pixels auf einer Grauskala liegt. Graue Farben zeichnen sich durch geringe Unterschiede zwischen den einzelnen Farbwerten aus. In der Praxis hat sich diese Farberkennung leider als nicht nützlich erwiesen.

Abbildung 2.5: Beispiel Farberkennung

Ein gutes Beispiel, wieso die Farberkennung der Analyse nicht hilft, ist Abbildung 2.5. Dort ist zu sehen, dass der Ball zwar hell ist, aber nicht weiß und auch nicht grau, sondern braun-gräulich. Zwar könnten die Differenz- bzw. Farbwerte der Erkennung allgemeiner formuliert werden, dann sind sie jedoch zu allgemein, um den Badmintonball von anderen Objekten unterscheiden zu können.

2.3 Clustering

Mit den vorangegangenen Schritten ist es also nun möglich, die sich bewegenden Objekte (Ball, Spieler, Zuschauer, Schläger, etc) zu extrahieren, sodass auf dem Background Subtraction Video nur noch diese Objekte zu sehen sind. Dann werden entsprechend der angegebenen Parameter und Kriterien Blobs mit Lokation und Größe extrahiert. Nun gilt es, denn Ball-Blob von den anderen Blobs zu unterscheiden, um nur die Flugbahn als Datenpunkte zu haben.

Sam Careelmont hat dazu in seiner Masterarbeit über Schlagklassifikation im Badminton eine Lösung vorgeschlagen [13][1]. Sie besteht darin, alle Blobs bzw. Datenpunkte eines Frames einer (möglichen) Flugbahn zuzuordnen, einfach gesagt: ein Clustering Algorithmus, wie er auch in Machine Learning Anwendung findet. Nach Analyse des Videos existiert dann ein Array von möglichen Flugbahnen, darunter im besten Fall auch die wirkliche Flugbahn des Balls.

Trajectory Matching

Um die Flugbahnen zu konstruieren, wird pro Frame jeder Datenpunkt einer Flugbahn bzw. einem Cluster zugeordnet. Dazu wird ein Abstandsmaß benötigt, das definiert, wie weit ein Blob von einer Flugbahn entfernt ist. Schlussendlich soll der Blob der ihm nächsten Flugbahn zugeordnet werden.

Dieses Abstandsmaß eines Balls ist zunächst durch die euklidsche Distanz beider Punkte definiert. Dazu sei der letzte Datenpunkt der Flugbahn T und der in Frage kommende Blob B mit den Attributen $(x, y, area)$ gegeben. Daraus ergibt sich die Distanz-Formel:

$$D(T, B) = \sqrt{(T_x * B_x)^2 + (T_y * B_y)^2}$$

Sinnvoll ist es, zusammen mit der Distanz zweier Punkte ein Gleichheitsmaß einzubringen, definiert durch den Unterschied der Größe der beiden Punkte. Dies ist leicht möglich, da zu jedem Punkt auch das $size$ -Attribut gegeben ist. Aus der Kombination des Abstandsmaßes und des Gleichheitsmaßes ergibt sich die Formel

$$D(T, B) = (1 - w) * \sqrt{(T_x * B_x)^2 + (T_y * B_y)^2} + w * \sqrt{(T_(area) * B_(area))^2},$$

in der w eine Gewichtung darstellt, um entweder das Abstands- oder das Gleichheitsmaß zu priorisieren. Da in diesem Kontext der Abstand zweier Punkte am wichtigsten ist, wird im Weiteren das Gewicht $w = 0.1$ verwendet.

Nun kann anhand dieser Formel ein Blob nähsten bzw. gleichsten Flugbahn zugeordnet werden. Es muss jedoch eine Grenze geben, anhand welcher bestimmt wird ob ein Blob einer Flugbahn zugeordnet werden kann, oder ob eine neue Flugbahn erstellt werden muss.

[1] Das gesamte Unterkapitel 'Trajectory Matching' basiert komplett auf der Arbeit von Sam Careelmont. Es wird im weiteren für bessere Übersichtlichkeit auf Verweise zu der Quelle verzichtet.

Das wird durch eine Threshold-Variable umgesetzt, hier mit dem Wert 150. Überstreitet die berechnete Distanz und Gleichheit diesen Wert, wird der Blob keiner Flugbahn zugeordnet, sondern eine neue Flugbahn mit diesem Blob als erstem Datenpunkt erstellt.

Für den Fall, dass ein Spieler den Ball verdeckt, muss der Algorithmus trotzdem in der Lage sein, den Ball der richtigen Flugbahn zuzuordnen. Eine Flugbahn zu 'schließen', also zu verhindern, dass ihr ein Blob zugeordnet werden kann, nachdem ihr in einem Frame kein Datenpunkt zugeordnet wurde, wäre also falsch. Dafür gibt es die Idle-Variable, ein Parameter, der aussagt, für wie viele Frames eine Flugbahn offen bleibt, bis entschieden wird, dass ihr wahrscheinlich nichts mehr zugeordnet wird und sie geschlossen wird.

Recursive Challenge Algorithmus

Bei dem bis jetzt definierten Trajectory-Matching Algorithmus kann es vorkommen, dass einer Flugbahn pro Frame zwei Blobs zugeordnet werden, da diese Flugbahn für beiden Datenpunkte die Näheste ist. Im Kontext der Anwendung macht dies aber keinen Sinn, da der Badminton-Ball pro Frame nur einmal enthalten ist. Dieses Verhalten verfälscht also das Ergebnis.

Um dies zu vermeiden wurde Sam Cleemont's Algorithmus um eine rekursive Komponente erweitert, die match-traj Funktion:

```
def match_traj(keypoint, list, threshold, frame_num, rec):
    if rec is True:
        keypoint = keypoint[0]
    max = [10000, -1]
    sub = False
    for ind, traj in enumerate(list):
        if traj[1] == frame_num:
            index = len(traj[0])
            dist = distance(keypoint, traj[0][index - 2][0])
            if dist < max:
                if distance(keypoint, traj[0][index - 2][0])
                < distance(traj[0][-1][0], traj[0][index - 2][0]):
                    max = [dist, ind]
                    sub = traj[0][-1]
        else:
            dist = distance(keypoint, traj[0][-1][0])
            if dist < max[0]:
                max = [dist, ind]
                sub = False
```

```
if max[0] > threshold:
    list.append([[[keypoint, frame_num]], frame_num])
    return list
else:
    list[max[1]][0].append([keypoint, frame_num])
    list[max[1]][1] = frame_num
    if sub is not False:
        list[max[1]][0].remove(sub)
        return match_traj(sub, list, threshold, frame_num, True)
    else:
        return list
```

Als Input-Parameter werden der Funktion der zuzuordnende Punkt, die momentan aktuelle Liste der Flugbahnen, der threshold sowie die momentane Frame-Nummer und die boolsche Variable rec übergeben, die indiziert, ob die Funktion regulär oder rekursiv aufgerufen wurde.

In einem for-Loop wird dann durch die Liste der Flugbahnen iteriert. Unterschieden wird zwischen den Fällen, ob der zu vergleichenden Flugbahn schon ein Blob zugeordnet wurde oder nicht.

In dem ersten Fall wird verglichen, ob der aktuelle Datenpunkt näher an der Flugbahn liegt als der schon zugeordnete. Ist das der Fall und der aktuelle Datenpunkt ist schlussendlich dieser Flugbahn am nächsten, wird der schon zugeordnete Blob aus der Flugbahn gelöscht und rekursiv einer neuen Flugbahn zugeordnet. Ist der schlussendlich nähesten Flugbahn für diesen Frame noch kein Datenpunkt zugeordnet, kann der Blob ohne weiteres der Flugbahn zugeordnet werden.

Max-Prinzip

Aus der Liste der Flugbahnen muss nun noch die richtige Flugbahn ausgewählt werden.

Das Konzept, dass dazu ausgewählt wurde, nimmt final nur die Flugbahn, der am meisten Blobs zugeordnet wurden. Die Annahme ist, dass zwar andere Flugbahnen zustande kommen, jedoch dank der Parameter-Einschränkungen die wahre Flugbahn die ist, der am meisten Blobs zugeordnet wurden.

Für die Aufnahmen der $Kamera_h$ hält diese Annahme, das Resultat soll später im Kapitel Ergebnisse genauer beschrieben werden. Durch die Probleme mit dem Flimmern bei $Kamera_v$ und deren Auswirkungen auf Background Subtraction und Blob Detection hält dort die Annahme nicht stand.

Durch das Flimmern wird zum Beispiel ein großer Teil einer Lampe an der Decke als bewegtes Objekt wahrgenommen. Da die Pixel nah beieinander liegen, wird diese Lampe dann fälschlicherweise als sich bewegender Blob identifiziert. Da das Flimmern konsistent ist, zieht sich dieser Fehler durch jeden Frame des Videos und wird als wahre Flugbahn mit den meisten Datenpunkten ausgegeben.

Ziel dieses Projekts ist es, einen Ballwechsel durch Aufnahmen zu visualisieren. Um dieses Ziel am Ende zu erreichen, es aber nicht möglich war die Aufnahmen zu wiederholen, ist der Schritt des 'Flugbahn Auswählens' für die Aufnahmen der $Kamera_v$ nicht automatisiert. Es wird, wie oben beschrieben Background Subtraction, Blob Detection und Clustering durchgeführt, dann wurde jedoch von Hand und nach Augenmaß die richtige Flugbahn ausgewählt. Vorgenannte Verfahrensweise ist nicht mehr notwendig, wenn die Aufnahmen beider Kameras fehlerfrei vorliegen, wurde aber hier angewendet, damit für dieses Projekt schlussendlich ein Ergebnis vorliegt.

Kapitel 3

Umsetzung

3.1 Framework Auswahl

OpenCV

Wie schon der Name verrät, ist die Open-Source Computer Vision Library, kurz: OpenCV, ein Framework, das bestimmte Funktionalitäten für Computer Vision und Machine Learning bereit stellt [14], dazu gehören unter anderem die Algorithmen für Background Subtraction und Blob Detection.

Durch die Tatsache, dass Open Source Projekte oft eine tolle Community für Hilfe haben und eine OpenCV Library für Python verfügbar ist, wurde dieses Framework ausgewählt und ein Großteil der theoretischen Schritte beruhen auf OpenCVs Funktionalitäten.

Für dieses Projekt wurde konkret OpenCV Version 3.4.2 benutzt.

Python, Numpy, PyTest

OpenCV stellt Interfaces für Java, C++ und Python bereit. Da die Entwicklerin am die meiste Erfahrung mit Python hat wurde Python gewählt, auch für die restliche Umsetzung des Software-Entwurfs. Den meisten Support für OpenCV 3.4 gibt und dies mit Python 2.7 zusammenhängt, wurde nicht die neueste Python Version (3.7), sondern Python 2.7 benutzt.

Für das Umgehen mit Bildern und Videos stellt OpenCV Bilder und Frames als eine Matrix von Pixeln dar. Um darauf zugreifen zu können und analysieren zu können, war vorallem die Python Library Numpy wichtig, die spezielle Funktionalitäten für Arrays und Matrizen/ n-dimensionale Arrays bereit stellt.

Um den Software-Entwurf entsprechend testen zu können, wurde außerdem das Python Testing-Framework PyTest benutzt.

Blender

Blender ist ein Open-Source Tool zur Animation von 3D Objekten [15]. Im Gegensatz zu Software wie Adobe Animate besitzt Blender ein Python Interface.

Das macht Blender zur besten Option für ein Visualisierungstool, da die vorhandenen Daten automatisiert eingelesen werden können und eine automatisierte Pipeline von Analyse bis Visualisierung geschaffen werden kann.

3.2 Spezifische Funktionalitäten

Die im wissenschaftlichen Teil aufgeführten Funktionalitäten wurden, wie dort beschrieben, im Software-Entwurf implementiert. Weiterhin gibt es Teile des Entwurfs, die nicht direkt mit diesen Methoden zusammenhängen, aber notwendig sind, z.B. der Umgang mit Daten.

Diese Funktionalitäten sollen im Folgenden genauer beschrieben werden.

3.2.1 Data Handling Klasse

Füllen leerer Frames

Wie im Kapitel Clustering beschrieben, kann es bei der Auswertung des Videomaterials passieren, dass in manchen Frames aus verschiedenen Gründen kein Ball erkannt werden sein. Dieses Problem muss spätestens für die 3D-Visualisierung behoben werden. Würde der Ball in diesen Frames in der Visualisierung einfach verschwinden, wäre das unangenehm für den Zuschauer.

Dieses Problem wird in der Data-Handling Klasse mit der Funktion datacleanse behoben. Hier ist das Prinzip, die leeren Frames mit Daten vor und nach dem nicht-Erkennen 'auszufüllen'. Ist die Lokation vor und nach dem Verschwinden bekannt, können die leeren Frames mit diesen Informationen gefüllt werden.

Dieses Vorgehen beschreibt nicht die wirkliche Flugbahn oder Geschwindigkeit, von konstanter Geschwindigkeit ausgegangen wird, was in der Realität jedoch nicht der Fall ist. Allerdings trägt es zur Verbesserung der Visualisierungsqualität bei und ermöglicht so eine bessere Darstellung des Ballwechsels.

Normalisierung Koordinatensystem

Die aus der Blob Detection gewonnenen Datenpunkte folgen einem festgelegten Koordinatensystem, beschrieben in Abbildung 3.1 Teil A und B.

Um die Flugbahn korrekt in Blender visualisieren zu können, müssen die Datenpunkte an ein anderes Koordinatensystem angepasst werden, wie es in Abbildung 3.1 Teil C zu sehen ist. Dazu werden nach dem Füllen der leeren Frames die Koordinaten der Ballwechsel so normalisiert, dass sie zu Blender passen:

x-Achse

normalisieren, sodass Ursprung an Spielfeld anliegt

$x' = x - 315$

y-Achse

Achse invertieren

$y' = y + 1080$

normalisieren, sodass Ursprung an Spielfeld anliegt

$y' = y - 315$

Kombination

$y' = y + 915$

z-Achse

Achse invertieren

$z' = z - 1920$

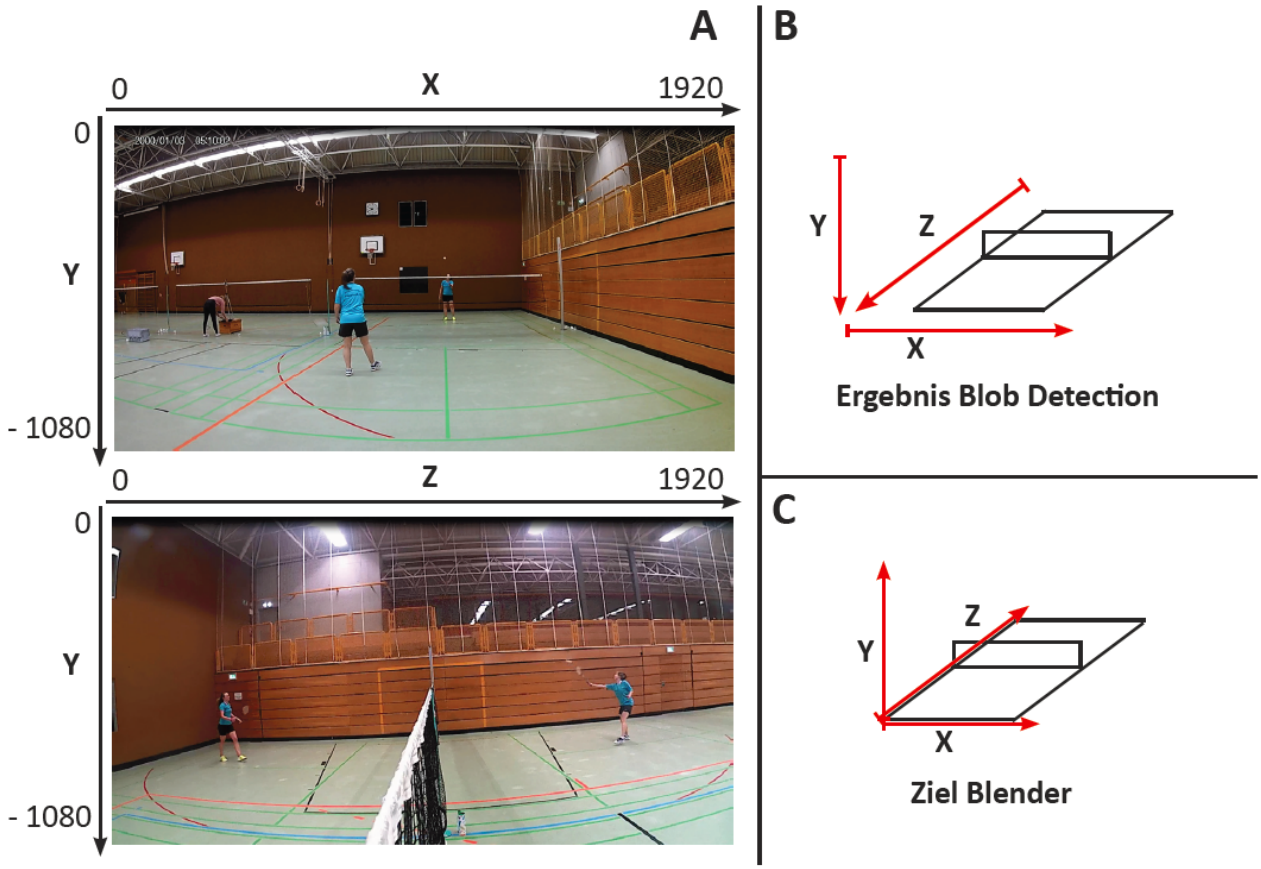

Abbildung 3.1: Normalisierung Koordinatensystem

Geschwindigkeitsbestimmung

Wie in der Aufgabenstellung beschrieben, ist es auch Teil der Aufgabe, die Geschwindigkeit einzelner Schläge zu bestimmen. Hier ist es wichtig zu beachten, dass für eine korrekte Geschwindigkeitsbestimmung nur die Daten aus der originalen Datei benutzt werden können. Leere Frames werden später unter der Annahme ausgefüllt, dass die Geschwindigkeit konstant ist, diese Werte reflektieren also nicht die reale Geschwindigkeit im Ballwechsel.

Um eine bessere Geschwindigkeitsbestimmung zu garantieren, wird zur Berechnung die zurückgelegte Strecke und Zeit über die letzten drei Frames benutzt.

```
counter = 0
for ind, row in enumerate(og):
        if row[1] != 0 and row[2] != 0:
            if counter > 1:
                distance = math.sqrt(abs(int(row[2]) - int(og[ind-2][2]))**2
                + abs(int(row[1]) - int(og[ind-2][1]))**2)
                distance_conv = distance * 0.0067
                # returns m/s
                speed = distance_conv / 0.032
                strip = new[ind].strip('\n')
                f.write(strip + ';' + str(int(speed)))
                f.write('\n')
            else:
                counter += 1
                f.write(new[ind])
        else:
            counter = 0
            f.write(new[ind])
```

Umgesetzt wird es durch eine Schleife, die durch die originale Datei `og` iteriert. Dabei wird die Bedingung gestellt, dass alle Koordinaten vorhanden sind, da hier noch nicht das Ausfüllen der leeren Frames durchgeführt wurde.

Sind diese Koordinaten in 3 Frames hintereinander vorhanden, wird die Geschwindigkeit berechnet. Dafür wird zuerst die Distanz über die letzten drei Frames berechnet, dann von Pixeln in Meter umgewandelt und die Distanz durch die Zeit geteilt (bei einer Framerate von 30 Frames entspricht ein Frame circa 0.0167 Sekunden, also die Zeit **zwischen** 3 Frames = 0.03). So wird errechnete Geschwindigkeit in die Datei geschrieben.

Klassifikation

Für die Klassifikation von Schlägen bzw. das Anzeigen eines neuen Schlages sind nur die Koordinaten der Dimension Z relevant, da diese die Schlagrichtung angeben.

Um einen Schlag zu beschreiben, wird eine Richtung gesetzt, die durch das Vorzeichen von δz definiert ist.

```
current = int(row[2]) - int(exi[ind-1][2])
if direction != math.copysign(1, current):
                direction = math.copysign(1, current)
                counter += 1
```

In der Umsetzung wird dafür das δz der momentanen Position und der Position vor einem Frame berechnet. Das Vorzeichen dessen wird dann mit dem Vorzeichen bzw. der Richtung des aktuellen Schlags verglichen. Unterscheidet sich dies wird ein neuer Schlag begonnen und die Richtung aktualisiert.

3.2.2 Blender Skript

Final müssen die gewonnenen und bereinigten Daten in Blender visualisiert werden.

```
frame_num = 0
for position in out2:
    bpy.context.scene.frame_set(frame_num)
    ob.location = [position[0], position[1], position[2]]
    print(ob.location)
    ob.keyframe_insert(data_path="location", index=-1)
    frame_num += 1
```

Zuerst werden die Koordinaten aus der Input-Datei gelesen. Dann wird das Objekt 'Ball' angesprochen, um pro Frame die Lokation anhand der Daten zu aktualisieren.

Weiterhin wird auf gleichem Weg die Nummer des Schlags und - wenn vorhanden - die Geschwindigkeit am Rand des finalen Videos angezeigt.

3.3 Programm Architektur

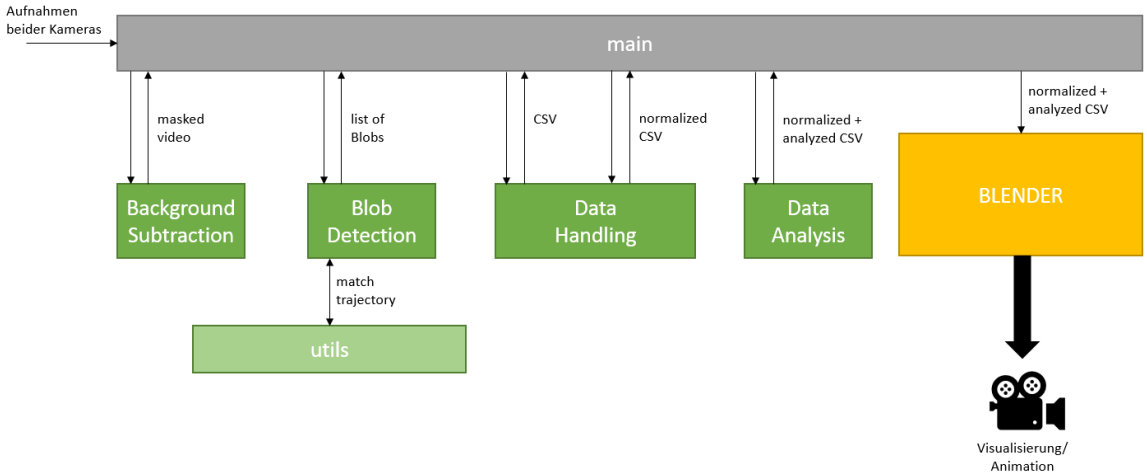

Abbildung 3.2: Architektur Programmentwurf

Abbildung 3.2 zeigt den Aufbau des vollendeten Programmentwurfs, in dem alle in Kapitel 2 beschriebenen wissenschaftlichen Methoden sowie die in Kapitel 3.2 beschriebenen speziellen Funktionalitäten enthalten sind.

Wie dort abgebildet, fängt der Prozess mit dem Input beider Aufnahmen an. Diese werden in der Background-Subtraction Klasse in zwei entsprechende Videos mit der oben beschriebenen Vordergrund- und Hintergrundmaske umgewandelt. Diese Videos werden dann an die Blob-Detection-Klasse übergeben, die zuerst das Video auf Blobs analysiert. Dann werden mithilfe der match-trajectory- und distance-Funktionen in der utils-Klasse die Blobs zu Flugbahnen kategorisiert und am Ende zwei Listen mit den Blobs beider Videos zurückgegeben.

Als nächstes wird zuerst die Blob-Liste der $Kamera_h$ an die Data-Handling Klasse gegeben, um dies in eine CSV-Datei zu schreiben und die Datenpunkte zu speichern. Dann wird die Blob-Liste der $Kamera_v$ ebenfalls an die Data-Handling Klasse gegeben und mithilfe der adddimension-Funktion zu der selben CSV-Datei hinzugefügt [1]. Es existiert nun nur noch eine Datei mit den Daten beider Kameras.

Diese Datei wird dann innerhalb der Data-Handling Klasse normalisiert und die Geschwindigkeit sowie Schlagklassifikation werden in der Data-Analysis Klasse ebenfalls der CSV-Datei hinzugefügt.

Die Daten können nun im CSV-Format an das Blender-Skript gegeben werden, um die Datenpunkte einzulesen und die Animation zu rendern.

Um gute Code-Qualität sicherzustellen wurde für dieses Projekt der Python Standard PEP8 angewendet.

3.4 Testing

Background Subtraction Klasse

Im Gegensatz zu beispielsweise mathematischen Funktionalitäten kann die Funktionsweise der hier vorliegenden Klassen und Funktionen nicht durch herkömmliche Tests verifiziert werden. So ist dies zum Beispiel bei der Background Subtraction Klasse der Fall, sie produziert ein Video. Hier wird zu statistischen Mitteln gegriffen, um Teile dieser Funktionalität zu testen.
In diesem Test wird das Originalvideo eingelesen und die Background Subtraction angewendet, danach wird das Output Video geöffnet. So kann verifiziert werden, dass ein Output in Videoformat existiert. Von diesem Video werden dann die Frames einzeln analysiert. Sie werden zuerst in Graustufenbilder umgewandelt und es werden dann die weißen Pixel in den Bildern gezählt. Der Test erfolgt dann, indem verifiziert wird, dass die Anzahl der weißen Pixel über 98% bzw. 50% liegt.

Dieser Test soll zeigen, dass die Background Subtraction funktioniert, indem angenommen wird, dass es nur einen kleinen Teil von bewegten Objekten gibt, der schwarz in dem Outputvideo auftaucht.
Für Aufnahmen der $Kamera_h$ kann der Testwert auf 98% gesetzt werden, für die Aufnahmen der anderen Kamera ist das nicht möglich. Man kann entweder den Test 'absichtlich' fehlschlagen lassen, um zu zeigen, dass es Probleme mit der Background Subtraction gibt. Um schlussendlich aber alle Tests erfolgreich durchlaufen lassen zu können wurde aber schlicht eine niedrigere Prozentzahl gewählt.

[1] Wie beschrieben war es im Verlauf des Projekts aufgrund der schlechten Kameraqualität nicht möglich, diesen Schritt anzuwenden, der Code dafür wird jedoch trotzdem bereitgestellt.

Blob Detection Klasse

Die Blob Detection Klasse soll anhand eines einfachen Erkennungs-Beispielbilds getestet werden. Es wurde ein Testbild so zusammengestellt, dass alle Blobs auf dem Bild mit den momentanen Parametern erkannt werden sollten. Auch die Blob Detection kann zwar nicht anhand eines Videos getestet werden, jedoch kann die korrekte Erkennung, sowie das Ausgabeformat getestet werden und es ist davon auszugehen, dass dies auch bei Videos wie erwartet funktioniert.

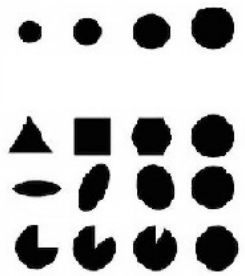

Abbildung 3.3: Testbild Blob Detection

Abbildung 3.3 zeigt das für den Test benutzte Beispielbild [12]; es wird erwartet, dass alle 16 Blobs erkannt werden. Dazu wird auf das Bild die Blob Detection angewendet und es wird dann die Länge der Keypoint-Liste mit der erwarteten Zahl verglichen.

Utils Funktionen

Um die match_trajectory-Funktion zu testen, wird ein einfaches Zuordnungsszenario mit einem Keypoint und einer Trajectory-Liste konstruiert. Dies wird der Funktion übergeben und eine neue Trajectory-Liste wird zurückgegeben.
Da Keypoints eine Klasse aus OpenCV sind und nicht direkt verglichen werden können, werden zuerst die x- und y-Werte aus der Rückgabeliste geparst und dann mit der erwarteten Liste verglichen. Dieser Test deckt durch die Auswahl des Szenarios auch die rekursive Funktionalität dieser Funktion ab und stellt sicher, dass dieser Vorgang erwartungskonform abläuft.

Die distance()-Funktion wird zwar indirekt durch eben erläuterten Test abgedeckt, wird aber gesondert mit einem einfachen Berechnungstest verifiziert, indem der Output mit dem erwarteten Ergebnis verglichen wird.

Datahandling Funktionen

Die writetocsv-Funktion wird getestet, indem eine einfache Flugbahn zum schreiben übergeben wird und dann mit einem vorbereiteten Array verglichen wird. Die Exceldatei wird innerhalb des Tests geöffnet, die Inhalte extrahiert und mit dem Array gleichgesetzt. Innerhalb dieses Prozesses wird durch das Parsen ebenfalls überprüft, ob sie den allgemeinen CSV Formaten entspricht.

Die datacleanse- und redim-Funktionen sind nur schwer zu testen, da die Ergebnisse in die Input-Datei geschrieben werden. Um zu vermeiden, dass der Test nur einmal durchgeführt werden kann, werden die Resultate der echten Videodateien auf ihre Richtigkeit überprüft.

Für den Test der datacleanse-Funktion wird die Datei von Video 1 innerhalb des Tests geöffnet und überprüft, ob in jeder Reihe alle drei Koordinaten vorhanden ist. So wird sichergestellt, dass alle leeren Frames gefüllt wurden.

Für den Test der redim-Funktion wird die gleiche Datei geöffnet; hier wird getestet, ob alle Werte innerhalb der erwarteten Dimensionen liegen. Für die x-Dimension ist das das Intervall [0, 1920], für die y-Dimension [0, 1080] und für die z-Dimension [-1920, 0].

Analysis Funktionen

Um schlussendlich auch die Funktionalitäten der Datenanalyse zu testen, kann innerhalb der geschriebenen Datei mehrmals getestet werden, da beide Funktionen robust genug gebaut sind um nur die Koordinaten zu lesen.

Sowohl für die Kategorisierung der Schläge als auch die Geschwindigkeitsberechnung werden die Funktionen auf einer Testdatei ausgeführt, dann wird diese eingelesen und mit einem Array, gefüllt mit den zu erwartenden Werten, verglichen.

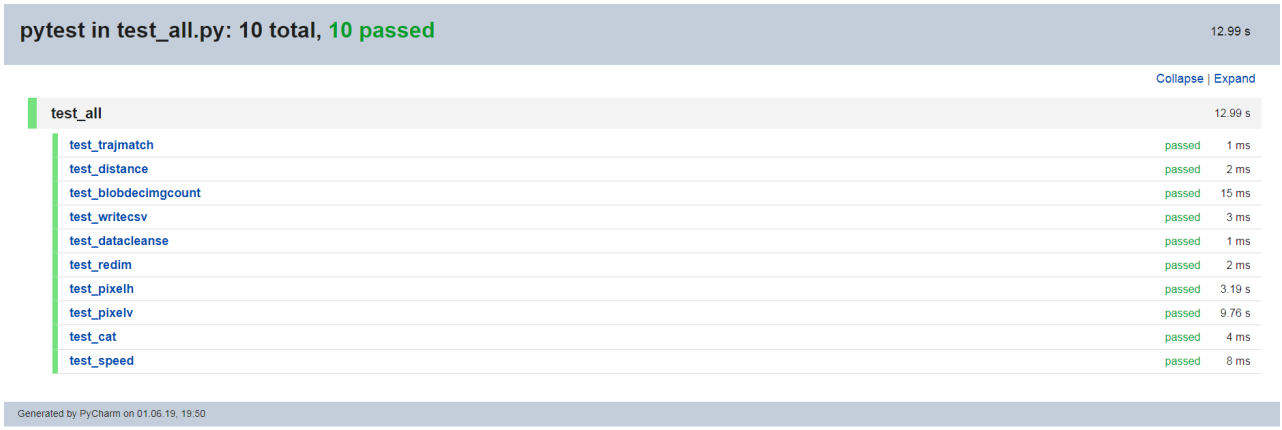

Abbildung 3.4: Output Py.Test

Wie in Abbildung 3.4 zu sehen ist laufen alle Tests fehlerfrei ab. Diese Übersicht ist eine Funktionalität der PyTest-Frameworks und kann, wenn nötig, exportiert werden.

Kapitel 4

Schluss

4.1 Ergebnisse

Am besten lässt sich das Ergebnis anhand eines konkreten Beispiels aufzeigen. Im Weiteren soll nun für das Beispiel-Aussschnit 1 der Vorgang und das Resultat gezeigt werden, da dieser Ballwechsel kurz und die einzelnen Schläge klar unterscheidbar sind, was die Darstellung in der Ergebnisbewertung übersichtlicher macht. Die Ergebnisse aller anderen Videoausschnitte befinden sich ebenfalls in dem Archiv-Ordner.

Zuerst wird die Background Subtraction angewendet, wie es in den Videos sync_1_1bgs.wmv und sync_2_1bgs.wmv zu sehen ist. Während die Background Subtraction in dem Video der $Kamera_h$ fast perfekt funktioniert und bis auf wenig Störflimmern nur die bewegten Objekte extrahiert werden, erscheint bei den Aufnahmen der anderen Dimension oft der Hintergrund als bewegtes Objekt, was die Aufnahme etwas verfälscht.

Abbildung 4.1: Beispiel nach Blob Detection und Clustering

Danach wird sowohl Blob Detection als auch zeitgleich das Clustering angewendet, das Ergebnis ist in Abbildung 4.1 zu sehen. Es zeigt sich klar die Flugbahn des Balls aus dieser Perpektive und bis auf einen Teil der Flugbahn nach dem Aufschlag wurde der komplette Ballwechsel mit den Blobs dargestellt.

Snippet	Resultat BackSub, BlobDec, Clustering	weiter benutzbar?	Fazit
1_1	+ + +	JA	6 von 8
1_2	+ +	JA	
1_3	-	NEIN	
1_4	+ +	JA	
1_5	+ + +	JA	
1_6	-	NEIN	
1_7	+ +	JA	
1_8	+	JA	
2_1	+	JA	6 von 8
2_2	+ +	JA	
2_3	-		
2_4	+	JA	
2_5	-	NEIN	
2_6	+		
2_7	++	JA	
2_8	++	JA	

Abbildung 4.2: Ergebnis nach Background Subtraction, Blob Detection und Clustering

Für das konkrete Beispiel aus Videosnippet 1 funktioniert Background Subtraction, Blob Detection und Clustering fast perfekt, Abbildung 4.2 zeigt die Ergebnisse nach diesen Schritten für alle vorhandenen Videoausschnitte [1].

Dort ist zu sehen, dass 6 von 8 ausgewerteten Ballwechseln für die Visualisierung zu gebrauchen sind. Für den Kontext des Projekts ist dieses Resultat als zufriedenstellend zu beurteilen und es ist anzunehmen, dass sich die Rate mit besseren Lichtverhältnissen und richtiger Kameraeinstellung nur noch verbessert. Das Ergebnis der Snippets 2_3 und 2_6 sind grau unterlegt, da die Ergebnisse von $Kamera_h$ nicht ausreichend gut sind und alle Dimensionen benötigt werden, um den Ballwechsel visualisieren zu können. So bleiben für den nächsten Schritt, die Visualisierung, nun die Videoausschnitte 1, 2, 3, 7 und 8.

Der weitere Prozess soll nun aber wieder anhand Beispielausschnitt 1 gezeigt werden. Abbildung 4.3 zeigt links die rohen Daten, wie sie aus dem vorherigen Schritt kommen. Auf der rechten Seite wurde dann Normalisierung und beide Analysen angewendet, die Lücken bzw. leeren Frames sind also ausgefüllt und die Anzahl der Schläge sowie - wo möglich - die Geschwindigkeit hinzugefügt.

Folgend werden dann diese Daten mithilfe des Blender Skripts visualisiert, wie im Archiv unter 'final visualization' im Video 1 zu sehen ist. Der Ballwechsel ist deutlich aus allen Pespektiven zu erkennen und wurde bis auf den fehlenden Teil nach dem Aufschlag perfekt aufgezeigt.

In Video 2 fehlt der größte Teil des Ballwechsels. Zwar ist der Teil des Ballwechsels, der vorne am Netz stattfindet, zu sehen, jedooch nicht der Aufschlag und der weite Schlag als Antwort auf den Aufschlag.

[1] +++ = der gesamte Ballwechsel wurde erkannt, ++ = der größte Teil des Ballwechsels wurde erkannt, + = Teile des Ballwechsels wurden erkannt, - = der Ballwechsel wurde gar nicht erkannt

39	954	-237	283			39	639	678	-1637	1	3
40	953	-240	277			40	638	675	-1643	1	2
41	953	-245	266			41	638	670	-1654	1	3
42	952	-249				42	637	666	-1663	1	
43	952	-249				43	637	666	-1672	1	
44	952	-254				44	637	661	-1681	1	
45	951	-258				45	636	657	-1690	1	
46	951	-261				46	636	654	-1699	1	
47	951	-265	213			47	636	650	-1707	1	
48	952	-272				48	637	643	-1715	1	
49	952	-272				49	637	643	-1723	1	
50	951	-276	189			50	636	639	-1731	1	
51	951	-281	184			51	636	634	-1736	1	
52	951	-286	180			52	636	629	-1740	1	2
53	951	-292	175			53	636	623	-1745	1	2
54	951	-292	171			54	636	623	-1749	1	2
55	952	-298	167			55	637	617	-1753	1	2
56	952	-303	164			56	637	612	-1756	1	2
57	952	-308				57	637	607	-1766	1	
58	951	-314	144			58	636	601	-1776	1	
59	951	-320				59	636	595	-1749	2	
60	951	-320				60	636	595	-1722	2	

Abbildung 4.3: Vorher und Nachher CSV-Datei

In Video 4 wird der Großteil des Ballwechsels dargestellt, sogar der schnelle Angriffsschlag (auch Smash genannt) in der Mitte der Ballwechsels. In diesem Video ist aber ein Fehler vorhanden, was bei circa 0:02 Sekunden zu sehen ist, wenn der Ball zurück in die rechte Feldhälfte fliegt und die Flugbahn einen Knick nach unten vollzieht. Betrachtet man die Dimensionen einzeln stimmt die Flugbahn mit der originalen überein. Es ist deshalb anzunehmen, dass dieser Ballwechsel durch einen nicht synchronen Anfang der beiden Videoausschnitte nicht korrekt dargestellt wird. Während in der y-Dimension der Ball schon auf dem Weg in die andere Feldhälfte ist, wird er in der z-Dimension erst geschlagen.

In Video 7 wird zwar der ganze Ballwechsel dargestellt, jedoch tritt in der Mitte dieses Ballwechsels ein Fehler auf. Während die Visualisierung korrekt den ersten Schlag nachvollzieht, bewegt sich danach der Ball in der rechten Feldhäfte für ein paar Sekunden vor und zurück. Vermutlich wurde dieser Fehler durch eine einzelne falsche Zuordnung eines Blobs verursacht, was durch das Ausfüllen der Lücke dann zu diesem Verhalten führt. Jedoch wurde der Schlag auf der rechten Seite korrekt dargestellt, da er auch in der Visualisierung dargestellt wurde; der Ball fliegt nur durch die falsche Zuordnung nicht korrekt auf die andere Feldseite. Der finale Schlag wird wieder korrekt dargestellt.

In Video 8 wurde fast der ganze Ballwechsel dargestellt, auch der Ballwechsel am Netz ist detailliert zu sehen. In dieser Visualisierung ist zu sehen, dass der Ball zwei Mal durch das Netz fliegt. Dies wird wahrscheinlich ebenfalls durch das Ausfüllen der leeren Frames verursacht. Wurde der Ball auf beiden Seiten des Netzes erfasst, wird die Flugbahn linear ausgefüllt, was ein solches Verhalten auslösen kann.

Die finale Visualisierung zeigt die Schwächen des Programmentwurfs. Durch die mangelhafte Blob-Erkennung fehlen kleine Teile der Ballwechsel. Zwar trägt das automatisierte Ausfüllen der Frames positiv zur Visualisierung dabei, verfälscht jedoch das Ergebnis und die Visualisierung ist teilweise nicht mehr originalgetreu.

Dennoch werden Großteile der Ballwechsel gut und detailliert dargestellt und die Flugbahn ist aus allen Dimensionen klar zu erkennen. Im Anhang wurden nur zwei Beispielperspektiven bereitgestellt, durch die dreidimensionale Natur der Daten kann aber jeder Blickwinkel ohne Erweiterung des Programms visualisiert werden.

4.2 Verbesserungen

Die offensichtlichste und auch am einfachsten umzusetzende Verbesserung ist die der Aufnahmequalität. Zwar können die Lichtverhältnisse nur begrenzt verändert werden. da je nach Halle nur wenig Beleuchtung vorhanden ist, doch schon mit etwas mehr Helligkeit während den Aufnahmen kann sich die Videoqualität signifikant verbessern. Des Weiteren macht es Sinn, für die Aufnahmen den 4K-Modus der Kameras zu aktivieren, um eine noch bessere Auflösung zu erreichen.

Die instensivste Berechnung ist die der Vorder- und Hintergrundmaske im Rahmen der Background Subtraction. Diese Berechnungszeit verschlechtert die Performance und es dauert momentan zu lange um diese Anwendung für Echtzeitanalysen zu benutzen. Da dies auf Matrizenrechnung basiert (jeder Frame ist eine Matrix aus Pixeln) könnte der Process durch diverse GPU Acceleration Frameworks beschleunigt werden.

Da die Action-Kameras, die für die Aufnahmen benutzt wurden, Weitwinkel-kameras sind, werden die Datenpunkte und Lokationen aus der Blob Detection verfälscht. Die 170 Grad Aufnahme verursacht, dass Objekte im Vergleich zur Realität gestaucht auf der Aufnahme zu sehen sich, je weiter sie von der Kamera entfernt sind. Damit die Flugbahn originalgetreu dargestellt werden kann, muss diesem Verhalten entgegengewirkt werden.

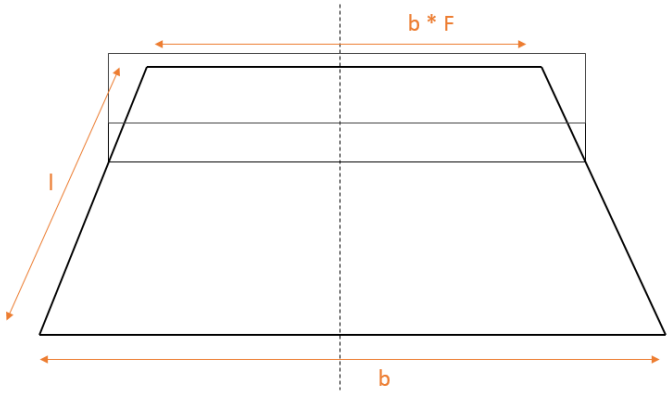

Abbildung 4.4: Darstellung Winkelkorrektur

Abbildung 4.4 verbildlicht die Problemstellung anhand der Perspektive von $Kamera_h$. Die Breite der Felds vorne b auf der Aufnahme unterscheidet sich um einen Faktor F von der Breite der hinteren Feldlinie $b * F$, wobei $F < 1$ ist. Um den Faktor F für eine beliebige Position p auf dem Feld zwischen der vorderen und hinteren Feldlinie zu berechnen, kann das Wissen über die Länge des Feld l und die Position des Balls auf dieser Geraden ausgenutzt werden:

$F_(pos) = pos * \frac{F}{l}$

Ist dieser Faktor bekannt, kann der Stauchung entgegengewirkt werden, indem die Position des Balls um F von der Mittellinie aus nach außen gezogen wird. Der Faktor F für die Aufnahmen von $Kamera_h$ wurde mit dem Wert 0.242 gemessen, für die Aufnahmen von $Kamera_v$ beträgt er 0.579.

4.3 Mögliche Erweiterungen

Im Laufe der Bearbeitung des Projekts sind viele Ideen zur funktionalen Erweiterung des Programms entstanden, die im Weiteren beschrieben werden sollen und die vielleicht die Grundlage für ein neues, fortführendes Projekt bieten könnten.

Eine Problemstellung, die bis jetzt noch nicht gelöst ist, ist die Frage, wann der Ballwechsel beginnt. Momentan sind die Videoausschnitte so gewählt, dass der Ballwechsel innerhalb der ersten halben Sekunde des Ausschnitts beginnt und es kann so die gesamte erkannte Flugbahn problemlos dargestellt werden. Gäbe es eine Möglichkeit den Anfang und das Ende eines Ballwechsels zu bestimmen, müsste die Videoaufnahme nicht mehr in einzelne Ballwechsel unterteilt werden und es könnte so eine ganze Spielsequenz analysiert werden, welches letzteendlich zur Qualitätsverbesserung des Programms beitragen könnte.

Eine weitere Funktionalität, die das Projekt weiter aufwertet, wäre die Information, ob ein Ball außerhalb des Feldes ist oder nicht. Diese Information gibt nicht nur dem Spieler bei der späteren Analyse zusätzliches Wissen über das Spiel, sondern würde das Programm näher an sein ursprüngliches Vorbild bringen - ein Hawkeye System.

Sind die Daten mit dem bestehenden Programm gesammelt, sind die Möglichkeiten zur Nach-Analyse endlos. Anhand der Flugbahn des Balls könnten die Schläge in die typischen Schlagkategorien im Badminton klassifiziert werden - Clear, Smash, Drop, etc. sowie Vor- und Rückhand. Daraus ergibt sich eine noch informationsreichere Nachbearbeitung des Spiels. Anhand dieser Daten können später Analysen der Spielstrategie erstellt werden. Dies ist schon mit einfachen Mitteln möglich, wie der Auszählung von Fehlern bei bestimmen Schlägen, kann aber hin bis zur Auswertung mit Machine Learning Methoden gehen, um dem Spieler neue Erkenntnisse über sein Spiel und seine Strategie zu ermöglichen.

Letztendlich wäre auch ein grafisches User-Interface nützlich und könnte das Programm zu einem Coaching-Tool machen. Hat man die Möglichkeit, dem Spieler in der Pause zwischen den Sätzen bestimmte Ballwechsel visuell aufzuzeigen (man wählt zum Beispiel aus einer Liste von Ballwechseln einen aus, der dann mit Ballwechselhistorie visualisiert wird)n kann der Spieler noch während des laufenden Spiels seine Fehler analysieren und seine Strategie anpassen. Erweitert man diese Idee um eine Nutzer-Funktion, in dem die Spiele nach Spieler klassifiziert werden können, eröffnet sich sogar die Möglichkeit eines Spielerprofils. Dokumentiert man über mehrere Turniere hinweg Spiele, erhöht sich die gesammelte Menge an Daten und es können sogar Fortschritte aufgezeigt werden.

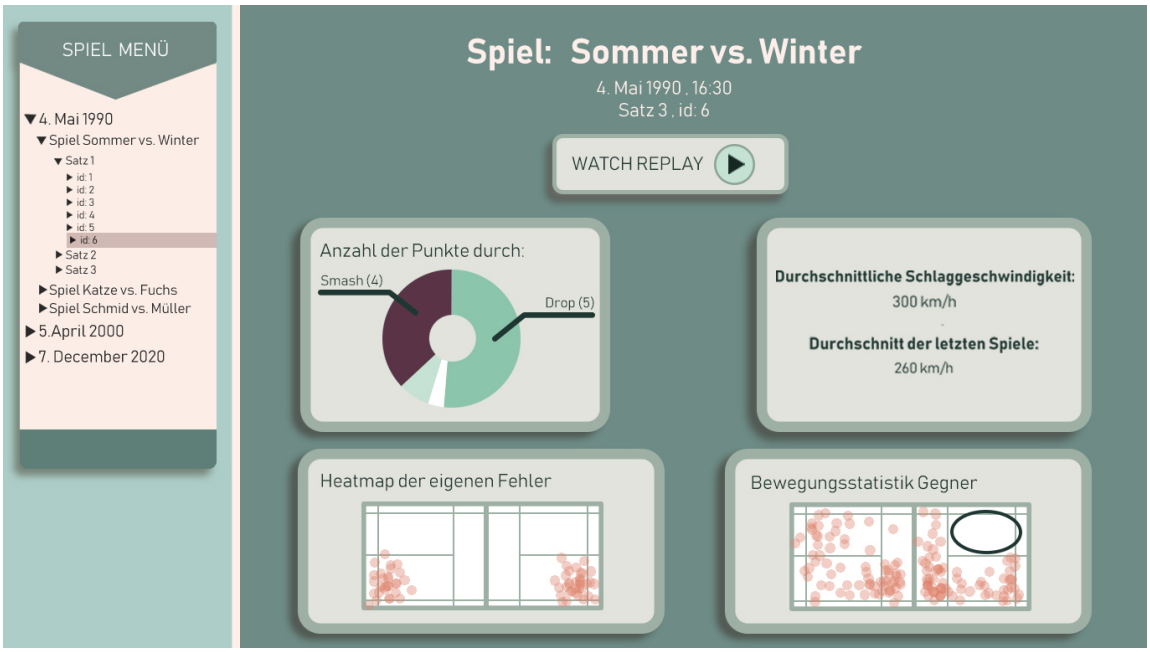

Abbildung 4.5: Mögliches User Interface für erweiterte Version

Abbildung 4.5 zeigt, wie das Programm, kombiniert mit den vorgenannten Erweiterungen, aussehen könnte. Links befindet sich eine Leiste, in der aus der Datenbank von Spielen und Visualisierungen ein Match ausgewählt werden kann. Dann werden auf dem Hauptscreen Informationen wie Datum und Spieler angezeigt und die Möglichkeit, sich diesen Ballwechsel Visualisieren zu lassen. Weiter unten könnten dann Statistiken zu dem gesamten Spielverlauf angezeigt werden, zum Beispiel, wie oben genannt, Klassifizierung von Schlägen, durch die Punkte erzielt wurden. Hat man erst ein Spielerprofil erstellt, können wie rechts oben durch die 'Schlaggeschwindigkeit' angedeutet, die Daten aus dem momentanen Spiel mit dem Durschnitt der letzten Spiele verglichen werden. Um besonders die Lokations- und Flugbahndaten auszunutzen, könnte man wie im unteren Teil dargestellt, beispielsweise sogenannte Heatmaps erstellen, die zeigen, wo sich der Gegenspieler während der Ballwechsel aufhält, beziehungsweise wo von dem Spieler selbst die meisten Fehler gemacht werden.

4.4 Fazit

Um das Gesamtkonzept wirklich für Amateurspieler zugänglich zu machen, bedarf es nur einiger Verbesserungen, die teilweise nicht mit der Software behoben werden können. So ist es vorallem wichtig, dass die Halle ausreichend beleuchtet ist, um gute Videoaufnahmen zu gewährleisten. Mit der Verbesserung der Videoqualität ergibt sich dann eine solide Dokumentiations- und Visualisierungssoftware, die mit handelsüblichen Kameras in einer für Amateurspieler üblichen Halle betrieben werden kann. Es bedarf außerdem keines technischen Wissens, es müssen lediglich die Aufnahmen gemacht werden, die Pipeline dieser Anwendung kann vollautomatisiert durchlaufen, ohne dass der Benutzer detailliertes technisches Wissen anwenden muss.

Im Rahmen dieser Studienarbeit wurde eine Möglichkeit gefunden, die Aufnahmen eines Badmintonspiels so zu analysieren, dass die Position und die Geschwindigkeit des Balls erkannt und dokumentiert werden kann. Es wurden drei wissenschaftliche Methoden vorgestellt und eingesetzt, um einzelne Schritte umzusetzen und Herausforderungen zu bewältigen.
Für diese Daten wurde ein Programm geschrieben, um mit ihnen umzugehen und sie zu analysieren.
Für die Visualisierung wurde mit Animationssoftware eine Szene nachgestellt, in der sich mithilfe der gespeicherten Daten der Ball wie in dem realen Ballwechsel bewegt und dessen Geschwindigkeit angezeigt wird.

Wie an der Vielzahl der Erweiterungsmöglichkeiten gut erkennbar, liefert schon das reine Konzept des Programms eine solide Grundlage für ein wirklich nützliches Visualisierungs- und später Analysetool im Badminton, das mit einfachen Erweiterungen auch für Amateurspieler zugänglich sein wird.

Literatur

[1] *Hawk-Eye Commercial Website*. https://www.hawkeyeinnovations.com/. Accessed: 2019-04-20.

[2] *IBM Blog: Watson powers US Open experiences for millions of tennis fans*. https://www.ibm.com/blogs/watson/2018/08/watson-powers-us-open-experiences-for-millions-of-tennis-fans/. Accessed: 2019-04-20.

[3] D. Gouwanda und S. M. N. A. Senanayake. "Emerging Trends of Body-Mounted Sensors in Sports and Human Gait Analysis". In: *4th Kuala Lumpur International Conference on Biomedical Engineering 2008*. Hrsg. von Noor Azuan Abu Osman u. a. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, S. 715–718.

[4] *Deutscher Badminton Verband: Badminton Spielregeln 2018/2019*. https://www.badminton.de/fileadmin/user_upload/18-dbv-druckwerk_satzung-ordnungen-spielregeln201819-website.pdf.pdf. Accessed: 2019-05-09.

[5] *Tan Boon Heong: der schnellste Spieler der Welt*. https://www.bzbasel.ch/sport/basel/tan-boon-heong-der-schnellste-spieler-der-welt-130138563. Accessed: 2019-05-11.

[6] Nir Friedman und Stuart J. Russell. "Image Segmentation in Video Sequences: A Probabilistic Approach". In: *CoRR* abs/1302.1539 (2013). arXiv: 1302.1539. URL: http://arxiv.org/abs/1302.1539.

[7] *OpenCV: Background Subtraction*. https://docs.opencv.org/3.4/db/d5c/tutorial_py_bg_subtraction.html. Accessed: 2019-05-27.

[8] *OpenCV: How to Use Background Subtraction Methods*. https://docs.opencv.org/3.4/d1/dc5/tutorial_background_subtraction.html. Accessed: 2019-05-27.

[9] Pakorn Kaewtrakulpong und Richard Bowden. "An Improved Adaptive Background Mixture Model for Realtime Tracking with Shadow Detection". In: *Proceedings of 2nd European Workshop on Advanced Video-Based Surveillance Systems; September 4, 2001; London, U.K* (Mai 2002). DOI: 10.1007/978-1-4615-0913-4_11.

[10] Zoran Zivkovic und Ferdinand van der Heijden. “Efficient adaptive density estimation per image pixel for the task of background subtraction”. In: *Pattern Recognition Letters* 27.7 (2006), S. 773–780. ISSN: 0167-8655. DOI: https://doi.org/10.1016/j.patrec.2005.11.005. URL: http://www.sciencedirect.com/science/article/pii/S0167865505003521.

[11] A. B. Godbehere, A. Matsukawa und K. Goldberg. “Visual tracking of human visitors under variable-lighting conditions for a responsive audio art installation”. In: *2012 American Control Conference (ACC)*. Juni 2012, S. 4305–4312. DOI: 10.1109/ACC.2012.6315174.

[12] *Blob Detection Using OpenCV*. https://www.learnopencv.com/blob-detection-using-opencv-python-c/. Accessed: 2019-05-19.

[13] Sam Careelmont. *Badminton shot classification in compressed video with baseline angled camera*. http://www2.imm.dtu.dk/pubdb/views/edoc_download.php/6575/pdf/imm6575.pdf. Accessed: 2019-05-28.

[14] *OpenCV: About*. https://opencv.org/about/. Accessed: 2019-05-18.

[15] *Blender: About*. https://www.blender.org/about. Accessed: 2019-05-18.