

# Optisches Spielertracking basierend auf mehreren Kameras

Bachelorthesis

**Michel Utz**  
Version 1.8 vom 17.01.2019

Studiengang:

BSc Informatik

Betreuer:

Marcus Hudritsch

Experte:

Harald Studer

# Inhaltsverzeichnis

Bachelorthesis	1
Inhaltsverzeichnis	2
1 Management Summary	4
2 Ausgangslage	5
2.1 Aufgabenstellung	5
2.2 Auftraggeber	5
2.3 Rollen und beteiligte Personen	5
3 Idee und Ziele	6
3.1 Performance	6
3.2 Kosten	6
3.3 Genauigkeit	6
3.4 Plattformunabhängigkeit	6
3.5 Einfache Konfiguration	6
4 Vorgehen	7
4.1 Aufgabenverwaltung	7
4.2 Zeitplan und Meilensteine	7
4.3 Arbeitsjournal und aufgewendete Zeit	8
4.4 Fortschritte	8
5 Videodaten	9
5.1 Magglingen	9
5.2 Eigene Aufnahmen 1	9
5.3 Eigene Aufnahmen 2	10
6 Lösung	11
6.1 Übersicht	11
6.2 Auslesen von Bildern aus Video	12
6.3 Personenerkennung	12
6.3.1 YOLO	13
6.3.2 Ablauf	13
6.4 Spielerextraktion	14
6.4.1 Filter	14
6.4.2 Schneiden der Ergebnisse	14
6.4.3 Farberkennung	15
6.4.4 Spielernummer	16
6.5 Positionsbestimmung	19
6.5.1 Perspektiven- und Modellsystem	20
6.5.2 Grundsätzliches Konzept	22
6.5.3 Referenzpunktauswahl – Dreieckslösung	24
6.5.4 Referenzpunktauswahl – Sonderfalllösung	25
6.5.5 Baryzentrische Berechnung	26
6.6 Tracking-Modul	27
6.6.1 Initiales Erkennen der Spieler	28
6.6.2 Wiederfinden der bestehenden Spieler	29
6.6.3 Bevorzugung aktueller Spieler	29
6.7 Korrekturfunktion	30
6.8 Weitere Funktionen	30
6.8.1 Berechnung der Laufweglänge	30
6.8.2 Logging	31
6.8.3 Debug-Informationen und Bilder	31
6.8.4 Tracking-Video exportieren	31
6.9 Tools	31
6.9.1 Groundtruth	31

6.9.2 Playerimageorganizer	31
6.9.3 Position	31
6.9.4 VideoCreator	31
6.10 Performance-Messungen	33
6.11 Performance-Verbesserungen YOLO	33
6.11.1 Hardware	34
6.11.2 YOLO-Konfiguration	34
6.11.3 TinyYolo	34
6.11.4 Anpassung von Target und Backend	34
6.11.5 OpenVino	35
6.11.6 Multithreading	35
6.12 Verbesserungen	35
6.12.1 Implementierte Verbesserungen	36
6.12.2 Verworfene oder zurückgestellte Verbesserungen	37
6.12.3 Nicht angegangene Verbesserungen	38
6.13 Entscheidungen	39
6.14 Fehlerfälle im Endprodukt	39
6.14.1 Bilder auslesen	39
6.14.2 Personenerkennung	40
6.14.3 Spielerextraktion	42
6.14.4 Positionsbestimmung	42
6.15 Logische Fehler im Tracking	44
7 Erfüllung der Ziele	56
7.1 Performance	56
7.2 Kosten	56
7.3 Genauigkeit	56
7.4 Plattformunabhängigkeit	56
7.5 Einfache Konfiguration	56
8 Fazit	57
9 Anhang	58
9.1 Benutzerdokumentation	58
9.1.1 Datenablage	58
9.1.2 Verwendete Libraries und Tools	58
9.1.3 Installation der Libraries	58
9.1.4 Ausmessen der Halle	60
9.1.5 Videoaufnahmen	61
9.1.6 Konfiguration im Code	61
9.1.7 Ausführen	62
9.1.8 Verstehen der angezeigten Bilder	62
9.1.9 Korrekturfunktion	67
9.1.10 Tools verwenden	67
9.2 Abbildungsverzeichnis	71
9.3 Tabellenverzeichnis	73
9.4 Glossar	73
9.5 Quellenverzeichnis	75
9.6 Arbeitsjournal	76
9.7 Code	76
10 Versionskontrolle	77
11 Selbstständigkeitserklärung	78

# 1 Management Summary

Diese Arbeit wurde im Rahmen der Abschlussarbeit (Bachelorthesis) des Studiengangs Informatik mit der Vertiefung «Computer Perception and Virtual Reality» an der Berner Fachhochschule erstellt.

Ziel war es, ein kostengünstiges System zur Berechnung und Verfolgung von Spielerpositionen zu erstellen. Basierend auf diesem System sind statistische Auswertungen denkbar und teilweise gemacht worden, welche als Zusatzinformation bei Fernsehübertragungen angezeigt oder zur Verbesserung der Mannschafts- oder Einzelleistung im Fussball verwendet werden können. Denn der moderne Fussball verlässt sich immer mehr auf die Informatik, um Spiele zu analysieren, sowie Auswertungen und Statistiken zu erstellen. Die durch Analysen gefundenen Informationen lassen zu, dass speziell zugeschnittene Trainingspläne zur Verbesserung der Leistung von Spieler oder Mannschaft erstellt werden [1][2]. Bestehende Systeme sind aber nicht kostengünstig und für kleinere Vereine und Privatpersonen nicht einfach finanziert. Meistens verlassen sich die aktuell verwendeten Systeme auf GPS, um die Spieler zu tracken.

Technologisch gesehen erleben die visuelle Bilderkennung und -verarbeitung in letzter Zeit einen grossen Aufschwung. Der Aufschwung gründet auch im Hype um die Neuronalen Netze, welche für solche Probleme vermehrt eingesetzt werden. So soll mit dieser Arbeit auch das neugefundene Potential eines visuellen Systems aufgezeigt werden.

In diesem Projekt wurde mit einem Budget von unter 1000 CHF ein System entwickelt, welches ein Spiel auf visueller Basis analysiert. Spieler werden erkannt und verfolgt. Das Aufsetzen des Systems zur Unterstützung eines neuen Feldes und für die Bearbeitung neuer Aufnahmen benötigt einige kleine Konfigurationen, welche von einer geübten Person in überschaubarer Zeit durchgeführt werden können. Die Resultate des Systems stimmen für ein neunzig Sekunden dauerndes Testsegment mit einer Akzeptanzdistanz von 2m zu 96% mit der Realität überein. Voraussetzung für diese Zahl ist, dass das Programm von einem Operator überwacht wird, welcher von Zeit zu Zeit korrigierend einwirken kann. Es bestehen zurzeit noch Performanceprobleme, welche eine Echtzeitauswertung verhindern. Diese Probleme konnten während des Projekts noch nicht behoben werden, sind aber mit etwas mehr Aufwand durchaus lösbar. Dafür wurden Lösungsansätze vorgeschlagen und teilweise getestet.

Die Applikation ist plattformunabhängig und kann dementsprechend auf einer breiten Palette von Betriebssystemen verwendet werden. Eine umfangreiche Benutzerdokumentation ermöglicht das Verwenden der Applikation für eigene Aufnahmen und erklärt Konfiguration, Anwendung und Output.

## Sprachliche Gleichstellung

Aus Gründen der besseren Lesbarkeit wird auf die gleichzeitige Verwendung männlicher und weiblicher Sprachformen verzichtet. Sämtliche Personenbezeichnungen gelten für beide Geschlechter.

## 2 Ausgangslage

### 2.1 Aufgabenstellung

Professionelle Fussballspiele werden mit mehreren Kameras gefilmt, die das Geschehen aus verschiedenen Blickwinkeln aufzeichnen. Ähnliche Möglichkeiten stehen bei Amateurspielen und Spielen unter Freunden selten zur Verfügung. Werden aber günstige Kameras eingesetzt, können von einem solchen Spiel auch mehrere Videos zur Verfügung stehen. Auf dieser Idee beruht die Datenbasis dieser Bachelorthesis: Über 30 Minuten Hallenfussball, gefilmt aus drei Perspektiven. Die Grundidee ist, auf Basis der genannten selbsterstellten Videos alle Spieler eines Fussballspiels zu identifizieren und zu verfolgen (Tracking). Spieler sollen, idealerweise in Echtzeit, eindeutig markiert und unterschieden werden können. Konkret könnten sie nach der Positionsbestimmung auf einem Fussballfeld-Modell angezeigt und so die Ergebnisse sichtbar gemacht werden. Das Tracking soll auf alle drei verfügbaren Videoperspektiven zurückgreifen, um Informationen zu gewinnen.

Mit dieser Arbeit ist die Grundlage für diverse Erweiterungen gegeben. Denn sobald dieser Teil der Lösung bekannt und implementiert ist, ist die Position jedes Spielers auf dem Platz zu jedem Zeitpunkt bekannt und es können anschliessend Statistiken erfasst oder die Bewegungen eines Spielers nachvollzogen werden.

Während der vorangehenden Arbeit, dem Projekt 2, wurde bereits eine Applikation entwickelt, welche für die Kameraaufnahmen versucht, die Spieler zu erkennen und Eigenschaften zu extrahieren. Somit besteht vor Projektbeginn eine Basis, die jedoch umgeschrieben und für die Verwendung von mehreren Kameras erweitert werden muss. Zusätzlich bestehen aktuell weder Verarbeitungsprozess noch Tracking-Algorithmus.

### 2.2 Auftraggeber

Da das Projekt aus einer eigenen Idee stammt, existiert kein bestimmter Auftraggeber. Die Rolle wird von der Projektleitung übernommen.

### 2.3 Rollen und beteiligte Personen

Es waren folgende Personen mit bestimmten Rollen am Projekt beteiligt.

Name	Rollen
Michel Utz	Projektleitung, Entwickler, Tester, Reviewer, Auftraggeber
Marcus Hudritsch	Betreuer, Begleitender Professor, Experte
Harald Studer	Experte

Tabelle 1 An dem Projekt beteiligte Personen und ihre Rollen.

## 3 Idee und Ziele

Die Idee, Spieler auf einem Spielfeld zu tracken ist nicht besonders neu. Bereits an der Fussballeuropameisterschaft 2008 kamen erste solche Systeme zum Einsatz [3]. Dort wurden 16 stereoskopische Kameras verwendet, welche mit 25fps die Spieler (und den Ball) aufnahmen. Laufwege und Distanzen wurden anschliessend ausgerechnet. Fehler wurden von einem Operator korrigiert. Für dieses Projekt stehen weniger Kameras zur Verfügung, dafür kann auf einen grossen technologischen Fortschritt der letzten zehn Jahre zurückgegriffen werden.

Die Ziele, die für diese Arbeit gesetzt wurden, sind absichtlich ambitioniert. Das heisst jedoch nicht, dass sie nicht erreichbar sind. Es war im Vorfeld aufgrund des relativ grossen Umfangs der Arbeit schwierig abzuschätzen, was im Bereich des Möglichen liegt.

### 3.1 Performance

Die Auswertung soll in Echtzeit geschehen. Es müssen dabei nicht alle Frames der Videos für das Tracking verwendet werden. Pro Sekunde sollen für jede beteiligte Kamera zwei Frames ausgewertet werden. Im Gesamten soll also eine Verarbeitung von sechs Frames pro Sekunde geschehen.

### 3.2 Kosten

Die finanziellen Mittel sollen die Kosten für drei GoPro Hero 5 Black nicht übersteigen. Pro Kamera wird mit etwa 300 CHF gerechnet [4].

Die Kosten für die Entwicklung der Tracking-Applikation sind zu vernachlässigen, da diese im Rahmen des Informatik-Studiums geschieht.

### 3.3 Genauigkeit

Die Software soll über ein Testsegment von 90 Sekunden sechs Spieler verfolgen können. Die Abweichung zwischen Resultat und Realität darf dabei für eine Spielerposition zu einem bestimmten Zeitpunkt maximal zwei Meter betragen.

Es dürfen höchstens 10% der Spielerpositionen falsch sein.

Anhand einer zu erstellenden Groundtruth soll dies geprüft werden.

### 3.4 Plattformunabhängigkeit

Die Applikation soll auf Windows und Ubuntu funktionieren, um die grundsätzliche Portabilität zu zeigen.

### 3.5 Einfache Konfiguration

Die Konfiguration der Applikation für ein neues Feld oder für neue Kameras soll einfach sein. Konkret soll das initiale Einrichten der Applikation inklusive der Konfiguration von drei Kameras nicht mehr als vier Arbeitsstunden dauern.

Das erneute Konfigurieren von drei Kameras für ein bestehendes Spielfeld soll maximal eine halbe Stunde dauern.

## 4 Vorgehen

### 4.1 Aufgabenverwaltung

Die Aufgaben oder Tasks wurden grösstenteils im Gratis-Tool Trello verwaltet. Zu finden ist das Board unter folgender Adresse: <https://trello.com/b/KTr1rsai/thesis>

Zugriff wird durch die Projektleitung auf Anfrage gewährt.

### 4.2 Zeitplan und Meilensteine

Die Zeiterfassung und die Meilensteine wurden im Microsoft Excel verwaltet.

Datum	KW	Woche	Meilenstein	Erreicht in Woche	Beschreibung
10.09.2018	37	0			
17.09.2018	38	1	Planung vollständig und abgenommen	1	Die Planung ist erstellt, wurde mit Marcus Hudritsch angeschaut und von ihm abgenommen.
24.09.2018	39	2	Alle Kameras konfiguriert	2	Die Referenzpunkte des Spielfeldes müssen für jede Kamera konfiguriert werden. Mit diesem Meilenstein ist das für alle drei Kameras gemacht.
01.10.2018	40	3			
08.10.2018	41	4	Groundtruth vorhanden	4	Es muss eine Groundtruth bestehen, um während der Entwicklung Lösungen miteinander zu vergleichen.
15.10.2018	42	5			
22.10.2018	43	6			
29.10.2018	44	7			
05.11.2018	45	8	Erste Rundum-Lösung "Multikameratracking"	8	Der Prototyp aus dem Projekt 2 wurde erweitert, sodass er mit mehreren Kameras funktioniert. Zudem wurde eine initiale Tracking-Lösung implementiert.
12.11.2018	46	9	Verbesserungen erkannt, priorisiert und geplant	9	Die erste Lösung wird sicher noch nicht perfekt sein. Es wird also Zeit eingeplant, um die Lösung zu analysieren, zu verbessern, zu aufzunehmen, zu priorisieren und zu planen.
19.11.2018	47	10			
26.11.2018	48	11			
03.12.2018	49	12	Wichtigste Verbesserungen implementiert	12	Die hochpriorisierten Verbesserungen sollen implementiert sein.
10.12.2018	50	13			
17.12.2018	51	14			

24.12.2018	52	15	Erster Wurf Dokumentation abgenommen	15	Die Grundlagen der Dokumentation sind erledigt, mit Marcus Hudritsch abgesprochen und mögliche Änderungen geplant.
31.12.2018	1	16	Book: Freigabe	16	Die Freigabe des Textes für das Book ist erfolgt.
			Plakat: Abgabe	16	Das Plakat wurde abgegeben.
07.01.2019	2	17			
14.01.2019	3	18	Abgabe Dokumentation + Projekt	18	Die Dokumentation wurde dem Begleiter, dem Experten und dem Sekretariat abgegeben.
			Präsentation vorbereitet und gehalten	18	Die Präsentation wurde gehalten.
21.01.2019	4	19	Film erstellt und abgegeben	(19)	Meilenstein ist nach der Abgabe des Berichts und kann dementsprechend noch nicht als erreicht gelten.
28.01.2019	5	20			
04.02.2019	6	21	Verteidigung gehalten	(19)	Meilenstein ist nach der Abgabe des Berichts und kann dementsprechend noch nicht als erreicht gelten.

Tabelle 2 Zeitplan und Meilensteine.

#### 4.3 Arbeitsjournal und aufgewendete Zeit

Es wurde während des gesamten Projekts ein detailliertes Arbeitsjournal geführt. Dieses ist im Anhang zu finden.

Da die Bachelorthesis 12 ECTS-Punkten entspricht, müssen ungefähr 360 Stunden aufgewendet werden [32]. Die Zeiterfassung geschah direkt im Arbeitsjournal.

Es wurden bis am 14.01.2019 insgesamt 376 Stunden aufgewendet. Das beinhaltet weder das Halten der Präsentation noch das Vorbereiten der Verteidigung. Auch das Erstellen des obligatorischen Films ist nicht enthalten.

#### 4.4 Fortschritte

Die Arbeit konnte zu Beginn des Projektes einfach geplant werden, da klar war, was für die erste Version des Programms benötigt wird. Sobald die grundsätzliche Umsetzung implementiert wurde, gab es viele Verbesserungsmöglichkeiten, welche priorisiert abgearbeitet wurden. Trotzdem waren die Fortschritte in diesem Teil des Projekts schwerer zu erreichen, da eine Änderung in einem Bereich der Applikation ein Problem zu einem Zeitpunkt lösen und ein anderes zu einem späteren Zeitpunkt hervorbringen konnte. Zudem gab es durchwegs wichtige Entscheidungen betreffend Algorithmus und Architektur zu treffen. Da diese Entscheidungen aufeinander aufbauten, konnte nicht gut abgeschätzt werden, ob eine andere Option eigentlich die bessere Lösung gewesen sein könnte.

Trotzdem war es die richtige Entscheidung, iterativ vorzugehen.

## 5 Videodaten

Im Verlaufe des Projektes und Vorprojektes wurden mehrere Videoaufnahmen geplant und ausgeführt. Jede Aufnahme hat spezifische Eigenschaften.

### 5.1 Magglingen



Abbildung 1 Beispiele für die Videodaten.

Die initialen Videodaten stammen aus dem Projekt 2 (Vorprojekt) und wurden in Magglingen aus einem bestehenden Mehrkamera-Trackingsystem extrahiert. Die Daten wurden von diesem System jedoch unkomprimiert gespeichert, was die Datenmenge bereits für wenige Sekunden Video sehr gross werden liess. Zudem waren die Videos nicht hochauflösend, wurden vom System wieder überschrieben und waren teilweise korrupt.

Die Aufnahmen waren aufgrund des Ortes und dem limitierten Zugang zu Halle/System (Zugang nur mit Schlüssel möglich) aufwendig. Auch widersprach eine fix montierte Videoanlage dem eigentlichen Sinn des Projektes. Somit wurde beschlossen, eigene Aufnahmen zu machen, um Zeit zu sparen und Kameraperspektive und andere Faktoren selbst kontrollieren zu können.

Aufnahmemeinformationen	
Datum	09.02.2018
Anzahl Kameras	4 wurden verwendet, zusätzliche wären verfügbar gewesen
Bildauflösung	1280x1024
FPS	25
Dauer der verfügbaren Videos	Etwa drei Minuten aufgezeichnet, Daten jedoch korrupt
Art der Aufnahme	Gestelltes Fussballspiel und gestellte Szenarien
Hilfsmittel Spielererkennung	Leichtathletik-Startleibchen

Tabelle 3 Aufnahmemeinformationen Magglingen.

### 5.2 Eigene Aufnahmen 1



Abbildung 2 Beispiel für die eigenen Videodaten.

Die ersten eigenen Aufnahmen entstanden während eines Trainingsspiels in einer Halle der Französischen Schule in Bern. Qualität und Dauer der Aufnahme konnten gegenüber den Aufnahmen in Magglingen deutlich verbessert werden. Auf diese Aufnahmen baut diese Thesis auf.

Aufnahmeinformationen	
Datum	22.03.2018
Anzahl Kameras	3
Bildauflösung	1920x1080
FPS	Zwei Kameras 60, eine Kamera 80
Dauer der verfügbaren Videos	~90 Minuten
Art der Aufnahme	Mehrere vollständige Fussballspiele
Hilfsmittel Spielererkennung	Ein Team hatte rote Trainingsleibchen mit aufgeklebten Nummern. Das andere Team hatte schwarze Oberteile teilweise mit und teilweise ohne Nummern

Tabelle 4 Aufnahmeinformationen der ersten eigenen Aufnahmen.

### 5.3 Eigene Aufnahmen 2



Abbildung 3 Beispiel für die verbesserten eigenen Videodaten.

Aufgrund von Erkenntnissen während des Projektes wurde beschlossen, eine weitere Aufnahme zu starten. Gründe dafür waren die schlechte Positionierung zweier Kameras, die entweder zu wenig des verfügbaren Bildes für das Feld aufgewendet hatten oder zu tief montiert wurden. Das führte in beiden Fällen zu einer schlechten Positionserkennung und vielen Falscherkennungen der Spielerdetektion. Dementsprechend war das Konzept der neuen Aufnahmen klar: Die Kameras mussten das Geschehen von weiter oben aufnehmen. Zudem gab es keine Kamera mehr, welche das gesamte Feld überblickte. Jede Kamera war für den Teil des Spielfeldes vorgesehen, den sie optimal überblicken konnte.

Aufnahmeinformationen	
Datum	22.11.2018
Anzahl Kameras	3
Bildauflösung	1920x1080
FPS	60
Dauer der verfügbaren Videos	~45 Minuten
Art der Aufnahme	Mehrere vollständige Fussballspiele
Hilfsmittel Spielererkennung	Ein Team hatte rote Trainingsleibchen mit aufgeklebten Nummern. Das andere Team hatte gelbe Oberteile mit aufgeklebten Nummern

Tabelle 5 Aufnahmeinformationen der zweiten eigenen Aufnahmen.

# 6 Lösung

Dieser Teil dokumentiert die finale Applikation, welche während der Bachelorthesis erstellt wurde. Zusätzlich wurden die wichtigsten Entscheidungen und Entwicklungsschritte hier festgehalten. Dieses Kapitel dokumentiert nicht den Umgang mit der Applikation. Für diesen Zweck besteht ein Benutzerhandbuch, welches im Anhang zu finden ist.

## 6.1 Übersicht

Mit untenstehender Grafik soll der grundlegende Ablauf für die Verarbeitung aufgezeigt werden. Dieser besteht aus diesen Schritten, welche in der Folge ausführlicher erklärt werden:

- Korrekturlogik
- Auslesen der Bilder aus den Videodateien
- Verarbeitung Bild Kamera 1 (Personenerkennung, Spielerextraktion und Positionsbestimmung)
- Verarbeitung Bild Kamera 2 (Personenerkennung, Spielerextraktion und Positionsbestimmung)
- Verarbeitung Bild Kamera 3 (Personenerkennung, Spielerextraktion und Positionsbestimmung)
- Tracking-Logik inklusive History (Wissen über die bisher erkannten Spieler)

Dieser Ablauf behandelt einen Zeitpunkt im Tracking und verarbeitet in einem Zyklus drei Bilder (eines pro Kamera). Der Algorithmus ist so konfiguriert, dass er pro Sekunde des Inputvideos zehn Zyklen verarbeitet. Das ergibt für drei Kameras eine Verarbeitung von dreissig Bildern pro Sekunde. Ausgehend von einem vollständigen Fussballspiel, welches 90 Minuten dauert, ergibt dies 162'000 Bilder, welche verarbeitet werden müssen. Wie lange die Applikation für diese Verarbeitung benötigt, wird in einem späteren Abschnitt behandelt.

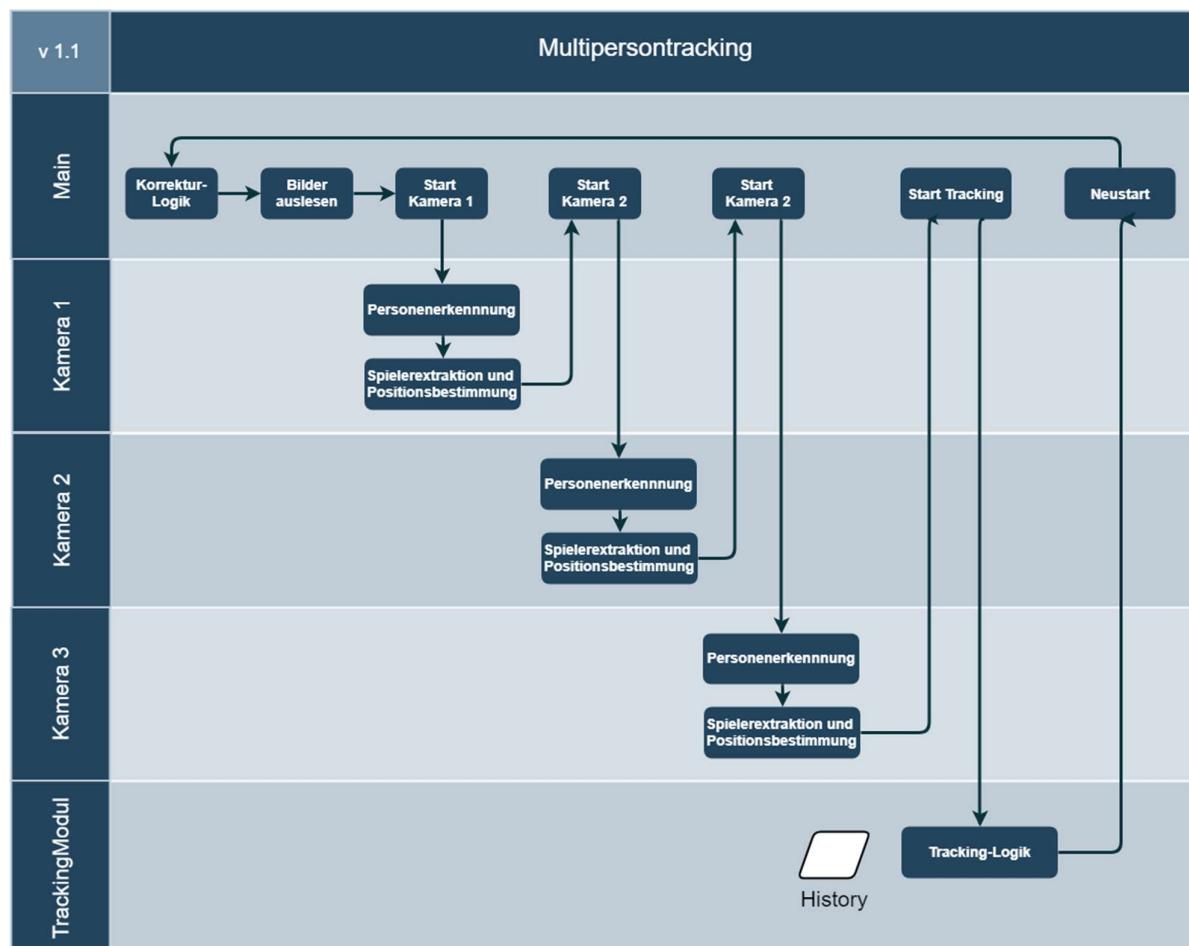


Abbildung 4: Grober Ablauf für die Verarbeitung der Bilder dreier Kameras.

## 6.2 Auslesen von Bildern aus Video

Pro Kamera wird eine Instanz der eigenen Klasse Camera benötigt, welche die jeweilige Videodatei kapselt. Mit einem Aufruf auf Camera::getNextFrame() wird das nächste Bild – welches in einer selbst definierten Zeitspanne liegen muss – zurückgeliefert. Um die Klasse für Videodateien mit verschiedenen FPS-Raten nutzbar zu machen, wird über die Videobilder iteriert und geprüft, ob die Anzahl Millisekunden seit Videostart für das aktuelle Bild in der gewünschten Zeitspanne liegt. Die Anzahl Millisekunden seit Videostart kann von der OpenCV-Klasse «VideoCapture» abgefragt werden [5]. Wenn das Frame nicht in der gewünschten Zeitspanne liegt, wird weiter iteriert. Wenn doch, wird das Bild zurückgeliefert und die neu gewünschte Zeitspanne erhöht. Im Fall, dass pro Sekunde zehn Bilder gelesen werden sollen, werden die Werte der gewünschten Zeitspanne um 100ms erhöht. So wird die FPS-Rate irrelevant für das Auslesen des richtigen Bildes.

In dieser Phase wird pro Kamera auf die beschriebene Weise je ein Bild ausgelesen und für die nächste Phase zwischengespeichert.



Abbildung 5: Ein Beispiel dreier ausgelesener Bilder (eines pro Kamera zum selben Zeitpunkt).

## 6.3 Personenerkennung

Aus jedem im letzten Schritt gewonnenen Bild werden nun durch einen Objekt-Detektor die Menschen detektiert.



Abbildung 6 Beispiel einer Personendetektion für ein Kamerabild.

In der obigen Abbildung ist eine beispielhafte Detektion abgebildet. Die grünen Rahmen umspannen ein detektiertes Objekt. Zudem hat jedes Objekt eine für dieses Bild eindeutige Nummer, welche als «Lokale ID» bezeichnet wird.

### 6.3.1 YOLO

«You only look once (YOLO) ist ein Detektionssystem, welches auf dem Neuronalen-Netz-Framework «Darknet» basiert. Dieses Open-Source-Projekt wurde für die Personenerkennung verwendet.

Viele Detektions-Systeme waren bisher darauf ausgelegt, Classifiers für die Detektion zu verwenden. So wird ein Model auf verschiedene Orte des Bildes angewendet. Regionen mit grosser Übereinstimmung werden als Treffer gewertet.

YOLO benutzt einen anderen Ansatz. Es wird ein einzelnes Neuronales Netz auf das ganze Bild angewendet. Dabei wird das ganze Bild in Regionen unterteilt und für jede Region eine Bounding Box inklusive Wahrscheinlichkeiten ermittelt. Gemäss den Entwicklern von YOLO bietet dieses Vorgehen verschiedenste Vorteile gegenüber dem ursprünglichen Ansatz der oben genannten Detektionssysteme. So wird bei der Detektion das gesamte Bild betrachtet, was dazu führt, dass die Vorhersagen in einem globalen Kontext unter Einbezug des ganzen Bildes getroffen werden. Zudem werden die Vorhersagen innerhalb eines einzelnen Netz-Durchlaufs gemacht, was Performancevorteile mit sich bringt [8][9].

Für dieses Projekt wird die Version 3 von YOLO verwendet, nachdem zuvor die Version 2 getestet wurde. YOLOv3 beinhaltet diverse Verbesserungen gegenüber den Vorgängerversionen, was bei einem Test klar sichtbar wurde. Sichtbar ist die Verbesserung in Abbildung 7. YOLO ist direkt in OpenCV integriert, was es zu einer naheliegenden Wahl für die Personenerkennung in diesem Projekt machte [10].



Abbildung 7 Vergleich der Detektion für ein Frame. YOLOv2 (links) erkennt deutlich weniger Spieler als YOLOv3 (rechts).

Es ist erwähnenswert, dass die YOLOv3-Implementation in OpenCV noch relativ jung ist und im Laufe des Projekts 2 auf unsere Anfrage hin implementiert wurde [11]. Das hat sicherlich eine grosse Hilfe dargestellt.

Verwendet wird durchwegs ein von den YOLO-Entwicklern vortrainiertes Netz, dessen Gewichtsdatei und Klassen heruntergeladen werden können. Mehr Informationen dazu sind im Benutzerhandbuch zu finden.

Die Integration von YOLO hat sich spät im Projekt als negativer Einfluss erwiesen. Denn auch wenn die Genauigkeit der Resultate die Anforderungen erfüllt, ist die Performance nicht ausreichend, um ein Echtzeit-Tracking umzusetzen. Dieses Thema wird in einem späteren Kapitel vertieft behandelt.

### 6.3.2 Ablauf

Um das Neuronale Netz zu erstellen, braucht es diverse Dateien und Parameter, welche hier beschrieben sind [6][7]:

- **Model:** Eine Datei mit der Endung «.weights». Beinhaltet die trainierten Gewichte des Neuronalen Netzes. Diese Datei kann direkt bezogen werden. Mehr Informationen dazu sind in der Benutzerdokumentation im Anhang zu finden.
- **Konfiguration:** Eine Datei mit der Endung «.cfg». Beinhaltet die Konfiguration des Neuronalen Netzes. Diese Datei kann direkt bezogen werden. Mehr Informationen dazu sind in der Benutzerdokumentation im Anhang zu finden.
- **Skalierung:** Ein Vorbearbeitungsparameter, welcher mit den Bildwerten multipliziert wird. Es wird der vorgeschlagene Wert 0.00392 übernommen.
- **Breite:** Ein Vorbearbeitungsparameter, welcher das Input-Bild auf eine gewisse Breite verkleinert. Der Standardwert ist 416 Pixel und wurde nicht geändert.

- **Höhe:** Ein Vorbearbeitungsparameter, welcher das Input-Bild auf eine gewisse Höhe verkleinert. Der Standardwert ist 416 Pixel und wurde nicht verändert.
- **Klassen:** Beinhaltet eine Liste von zu detektierenden Klassen. Diese Datei kann direkt bezogen werden. Mehr Informationen dazu sind in der Benutzerdokumentation im Anhang zu finden.

Sobald ein Netz erstellt wurde, kann ein Bild der Methode PlayerExtractor::getOuts() übergeben werden. Dieses verarbeitet das Bild zu einem BLOB und übergibt es dem Netz, welches eine Matrix zurückliefert. Darin befinden sich n detektierte Objekte sowie die dazu detektierten Klassen und die dazugehörigen Koordinaten (Zentrum, Höhe, Breite) [7].

## 6.4 Spielerextraktion

Als Parameter werden die detektierten Objekte aus der letzten Verarbeitungsstufe, das dazugehörige Kamerabild, eine Grössenschranke sowie die Referenzpunkte für die aktuelle Kamera übergeben. Diese Methode liefert für die Ergebnisse der Personenerkennung die dazugehörigen angereicherten Spielerobjekte.

### 6.4.1 Filter

Es werden während der Extraktion der Spielerobjekte diverse Resultate der YOLO-Personenerkennung herausgefiltert:

- Erkannte Objekte, welche nicht über der Sicherheits-Schranke des Neuronalen Netzes liegen, werden sofort verworfen. Das Netz war sich dort nicht genügend sicher, dass es wirklich eine Person ist. Der Standardwert der Schranke (50%) wurde nicht angepasst.
- Erkannte Objekte, welche vom Neuronalen Netz nicht das Label «Person» bekommen, werden ebenfalls verworfen, da sie für das Tracking uninteressant sind. Für Erweiterungen des Programmes könnte zusätzlich das Label «Sportball» interessant sein.
- Erkannte Objekte, welche kleiner als eine übergebene Grössenschranke sind, werden ebenfalls verworfen. Die Grössenschranke ist bei jedem Methodenaufruf (ein Aufruf pro Kamera pro Frame) mitzugeben, da die Perspektive der Kamera die möglichen Größen der Personen beeinflusst. Mit diesem Filter lässt sich erreichen, dass kleine Falscherkennungen nicht als Spieler erkannt werden.
- YOLO erkennt manchmal ein einzelnes Objekt als zwei Objekte. Diese Duplikate werden herausgefiltert, indem für jedes Objekt nochmals über die für das Kamerabild erkannten Objekte iteriert und eruiert wird, ob ein ähnliches bereits erfasst wurde. Der Vergleich der Ähnlichkeit wird mittels absoluter Distanz vollzogen. Objekte, die in der Summe 30 x- oder y-Pixel voneinander entfernt sind und dieselbe Farbe haben, werden als eines betrachtet. Sind die Punkte Obj1 und Obj2 gegeben, wird die Distanz wie folgt berechnet.

$$\text{Distanz} = | Obj1.x - Obj2.x | + | Obj1.y - Obj2.y |$$

### 6.4.2 Schneiden der Ergebnisse

In diesem Schritt werden die Spieler aus dem vollständigen Kamerabild ausgeschnitten, sodass ein Bild entsteht, auf dem idealerweise nur noch der Spieler zu sehen ist. Dabei werden die vom Neuronalen Netz gelieferten Koordinaten (Zentrum, Höhe, Breite) des detektierten Objektes verwendet, um den damit eingeraumten Spieler in einem eigenen Bild zu speichern, welches für die Farb-, Positions- und Nummernerkennung gebraucht werden kann. Konkret werden für das Ausschneiden vier Werte benötigt: Die bereits bekannten Werte «Höhe» und «Breite» sowie die einfach zu berechnenden «left» und «top»

$$left = \text{Zentrum}.x - \frac{\text{Breite}}{2}$$

$$top = \text{Zentrum}.y - \frac{\text{Höhe}}{2}$$

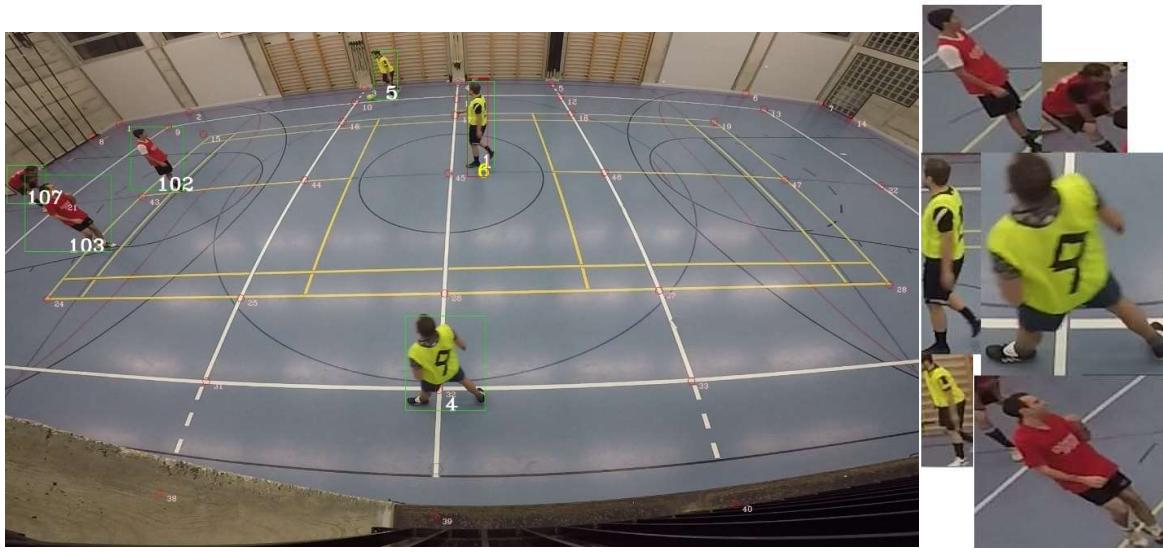


Abbildung 8 Links ist ein Teil eines Kamerabilds mit allen Spielern zu sehen. Rechts sind die sechs daraus gewonnenen Spielerbilder ersichtlich.

Die Ergebnisse von YOLO können in gewissen Fällen out-of-bounds sein. Das bedeutet, dass die zurückgelieferten Ergebnisse zu einem Fehler führen können, wenn die dazugehörigen vom Netz gelieferten Koordinaten benutzt werden, um einen Teil des Bildes zu extrahieren. Der Ausschnitt liegt nicht vollständig innerhalb des übergebenen Kamerabildes. Deswegen besteht eine Prüfung, welche die Ergebnisse bei Bedarf so zurechtschneidet, dass sie innerhalb des Bildes liegen.

#### 6.4.3 Farberkennung

Als Parameter nimmt die Farberkennung das wie im letzten Kapitel beschrieben ausgeschnittene Spielerbild entgegen. Zudem kann mit einem Boolean bestimmt werden, ob es ein herkömmliches Bild oder ein Ausnahmefall ist. Diese Unterscheidung besteht, um bei den zweiten eigenen Aufnahmen die etwas dunkleren Bilder einer speziellen Kamera zu behandeln. Bei diesen Bildern funktioniert die hier beschriebene Methode nicht zuverlässig. Der Ausnahmefall kann in der Zukunft entfernt werden, es handelt sich um eine durch problematische Daten verursachte Notlösung.

Standardmäßig wird zwischen roten und nicht roten Spielern unterschieden. Das in der Folge beschriebene Vorgehen wird implementiert, um zu ermitteln, in welche Klasse ein Spielerbild fällt [12]:

1. Umwandlung des Spielerbildes in den HSV-Farbraum
2. Anpassen der Grösse des Spielerbildes auf 100x100 Pixel
3. Erstellen von zwei leeren Bildern A und B (je 100x100 Pixel)
4. Threshold X auf Spielerbild anwenden und Ergebnis in Bild A speichern
5. Threshold Y auf Spielerbild anwenden und Ergebnis in Bild B speichern
6. Anzahl weisse Pixel in Bild A und B berechnen und die Gesamtanzahl der Variable SUM zuweisen
7. SUM durch die totale Anzahl Pixel ( $2 \cdot 100 \cdot 100$ ) dividieren und Resultat der Variable RES zuweisen
8. Wenn RES grösser als 0.025 ist, wird der Spieler als rot betrachtet

Die beiden Schwellwerte X und Y sind zwei Farbbereiche mit unterschiedlichen Farbwerten (hue), welche vom menschlichen Auge beide als rot erkannt werden.

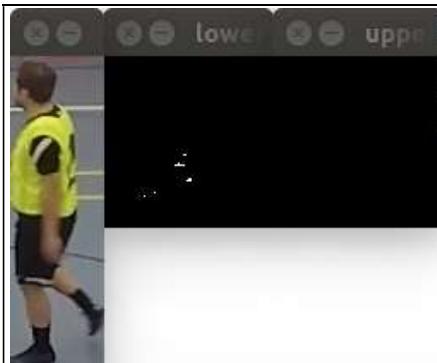


Abbildung 9 Ein gelber Spieler und die Resultate nach den Thresholds X (Mitte) und Y (rechts). Es wird nicht viel als rot erkannt.

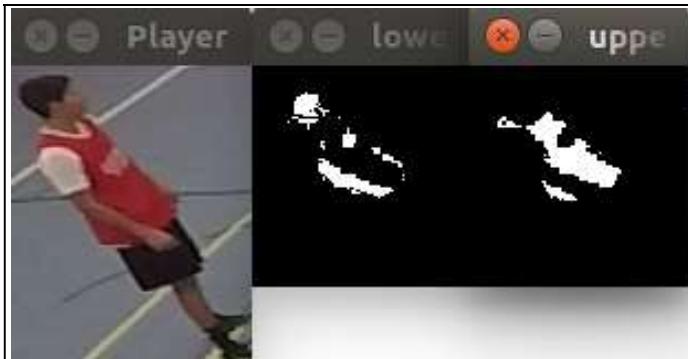


Abbildung 10 Ein roter Spieler und die Resultate nach den Thresholds X (Mitte) und Y (rechts). Im Gegensatz zu einem gelben Spieler überleben genug Pixel die Thresholds, sodass dieser Spieler als «rot» erkannt wird.

Im Ausnahmefall wird nicht auf rote, sondern auf gelbe Farbe geprüft. Danach wird die Summe der als gelb erkannten Pixel SUM2 mit der Summe der roten Pixel SUM verglichen. Die grössere Summe bestimmt die Farbe. Verschiedene Methoden sind ausprobiert worden und diese hat sich durchgesetzt, weil sie sich als am verlässlichsten herausgestellt hat.

Nach dem Projekt bliebe abzuklären, ob die Farberkennung konfigurierbar gemacht werden sollte. Denn für schlecht beleuchtete Aufnahmen könnte sich die bestehende Lösung als nicht genügend robust herausstellen.

#### 6.4.4 Spielernummer

Die initiale Nummererkennung aus dem Projekt 2 wurde initial übernommen. Dort wurden die Spielerbilder der eigenen Klasse NumberExtractor übergeben, welche mit SIFT nach Zahlen gesucht hat. Dabei wurden die Feature Points der T-Shirt-Fotos (mit den draufgeklebten Nummern) generiert und in den Bildausschnitten, welche die Spieler enthalten, gesucht. Die Bilder sind in der nachfolgenden Abbildung ersichtlich.



Abbildung 11 Beispiele der verwendeten T-Shirt-Fotos.

Es besteht die Problematik, dass die OpenCV Implementierung von SIFT sich als zu wenig robust gegenüber Größenveränderungen herausgestellt hat. Die Spielerbilder sind relativ klein und die Nummern nicht immer gut zu erkennen. Zudem ist es nicht gelungen, eine Anzahl übereinstimmender Feature Points als Schwellwert zu setzen, damit die Klassifizierung grösstenteils korrekt ausfällt. Konkret kann auf einem Bild, das überhaupt keine Nummer enthält, mit grosser Sicherheit eine beliebige Nummer erkannt werden. Zudem ist SIFT tendenziell zu langsam. Aus diesen Gründen wurde nach Alternativen gesucht.

Auf Hinweis von Herrn Hudritsch wurde Tesseract in Betracht gezogen und getestet. Dieses Tool bringt aber für die realen Daten keine Verbesserung im Vergleich zu SIFT.

Im Endeffekt wurde die Nummererkennung aus dem Projekt ausgebaut, da sie keinen Mehrwert liefert hat. Das führt zu Einschränkungen der Applikation. Bei Verwechslungen hat das Programm kein Hilfsmittel, um diese zu korrigieren. Bestünde eine Möglichkeit, eine Person mit hoher Sicherheit

zu erkennen, könnte die Detektion mit dem Tracking abgeglichen werden. Stimmt die Identität nicht mit der des getrackten Spielers überein, könnte eine Korrektur in Betracht gezogen werden.

Die Möglichkeit einer automatischen Identitätserkennung mit einem Neuronalen Netz wurde geprüft. Basierend auf den Bildern der ersten 50 Sekunden des Video-Testsegmentes werden die aus der YOLO-Personenerkennung resultierenden Spielerbilder gespeichert. Um keine zu ähnlichen Bilder zu erhalten, werden nicht pro Sekunde alle zehn Bilder pro Kamera gespeichert, sondern nur ein Frame pro Kamera pro Sekunde. Die so erhaltenen Bilder werden klassifiziert und können für das Training eines Neuronalen Netzes verwendet werden. Als Validierungsdaten werden ebenfalls Daten direkt exportiert. Alle Validierungs-Spielerbilder stammen aber von den letzten 28 Sekunden des Testsegmentes und überschneiden sich nicht mit den Trainingsdaten. Hier werden nur alle 1,5 Sekunden die Frames ausgewertet. Die Daten sind also unterschiedlich.



Abbildung 12 Beispielbilder aus dem Trainingsset. Alle sechs Klassen sind enthalten: "Aendu", "Daevu", "Michel", "Lucas", "Ishaqh" und "Kevin".

In der obenstehenden Grafik sind zur Veranschaulichung einige Beispiele der sechs Klassen dargestellt. Auf eine Darstellung der Validierungsbilder wird verzichtet, da diese ähnlich aussehen. Die Datensets setzen sich wie folgt zusammen.

Klasse	Anzahl Bilder
Aendu	197
Daevu	137
Ishaqh	179
Kevin	188
Lucas	193
Michel	189

Tabelle 6 Zusammensetzung des Trainings-Datensets.

Klasse	Anzahl Bilder
Aendu	20
Daevu	20
Ishaqh	24
Kevin	25
Lucas	23
Michel	15

Tabelle 7 Zusammensetzung des Validierungs-Datensets.

Umgesetzt ist ein Transfer-Learning. Dabei wird ein bestehendes und bereits trainiertes Neuronales Netz angepasst, damit es andere Klassen lernt. Dafür muss die letzte Schicht, welche die Klassifizierungen vornimmt, ersetzt werden. Anstelle der alten Schicht tritt eine neue, welche in diesem Fall sechs Klassen erlaubt. Diese Klassen haben die Namen "Aendu", "Daevu", "Michel", "Lucas", "Ishaqh" und "Kevin". Dies alles geschieht basierend auf einem von Matlab freigegebenen Skript [24]. Dabei wird das bereits trainiert zur Verfügung gestellte GoogLeNet verwendet [25]. Es wird als wichtig erachtet, dass die Validierungs- und Trainingsdaten strikt getrennt sind. Auf bereits beschriebene Art und Weise wird dies sichergestellt. Zusätzlich ist das Skript, welches standardmäßig einfach ein Datenset entgegennimmt und dieses in Validierungs- und Trainingsdaten aufteilt, korrigiert, damit es mit zwei separaten Datensets funktioniert.

Wie in der folgenden Abbildung sichtbar, sind die Ergebnisse überzeugend. Die dunkelblauen und schwarzen Punkte zeigen die Validierungsergebnisse für die Accuracy und den Loss. Trainiert wird auf einem Lenovo Yoga 720. Der Vorgang dauert 65 Minuten und erreicht eine Accuracy von 98.43%.

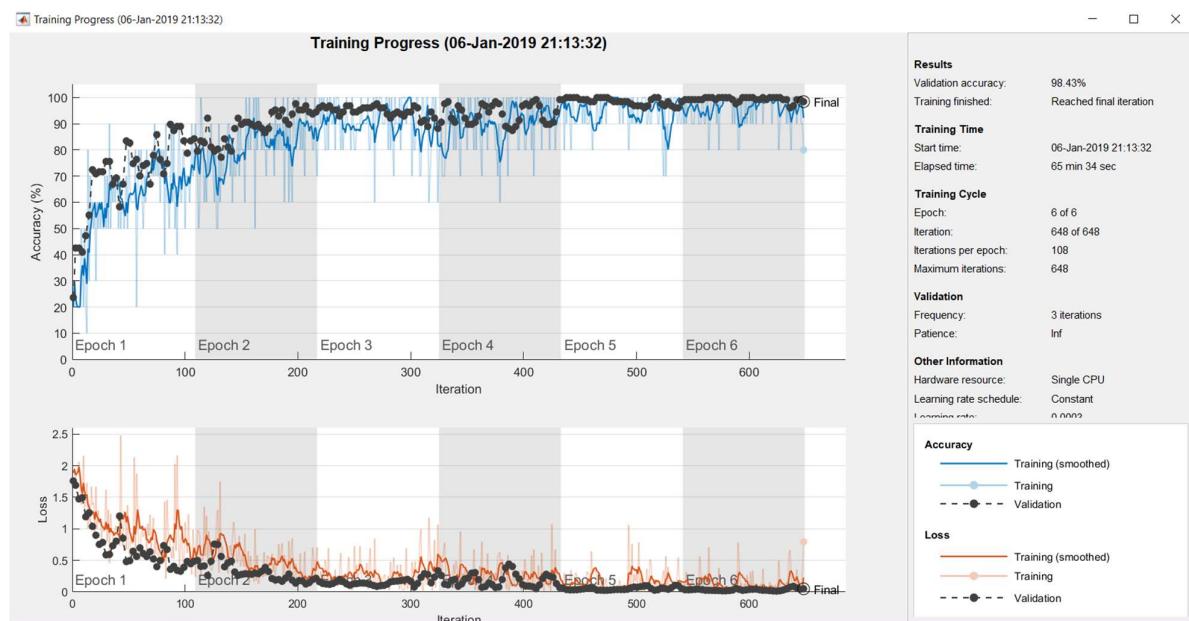


Abbildung 13 Der beste Durchlauf mit den Verläufen für Accuracy (oben, blau) und Loss (unten, rot).

Von den 127 Validierungsbildern werden nur zwei falsch klassifiziert. Die restlichen Bilder werden korrekt erkannt, was zum Teil erstaunlich ist, da die Bilder qualitativ nicht besonders gut sind. Bei den falsch klassifizierten Bildern wird ein Bild der Klasse «Daevu» als «Kevin» und ein «Lucas» als «Michel» klassifiziert. Die Verwechslungen sind verständlich, da die Trikotfarben der jeweiligen Spieler gleich sind. Das Netz ist sich zudem nur zu 71.8% und 79.2% sicher, dass es richtig liegt. In der folgenden Abbildung sind einige Beispiele klassifizierter Bilder des Validierungssets aufgezeigt.



Abbildung 14 Acht Beispielbilder, welche korrekt klassifiziert werden (links), sowie die zwei Falschklassifikationen (rechts).

Dieses Resultat zeigt, dass es grundsätzlich möglich ist, mit weniger als 200 Bildern eines Spielers eine sehr gute Klassifikation vorzunehmen. Implementiert ist zwar erst eine Lösung in Matlab, doch es wäre denkbar, dass das trainierte Model aus Matlab exportiert wird. Anschliessend könnte das Model, welches sich dann im ONXX-Format befindet, zu einem in OpenCV verwendbaren Dateityp umgewandelt und in OpenCV importiert werden [26, 27, 28]. Die Zeit dafür war leider nicht mehr vorhanden. Trotzdem soll an dieser Stelle eine zukünftige Lösung aufgezeigt werden. Dabei wird zwischen den Lösungen für eine Echtzeit- und eine Nachverarbeitungsapplikation unterscheiden.

Für eine Echtzeitapplikation bringt diese Lösung nur bedingte Vorteile, da die Klassifizierung zwar gut, aber etwas schwerfällig ist. Denn zuerst muss ein Model für die aktuellen Spieler trainiert werden, was Zeit in Anspruch nimmt. Anschliessend wäre zu prüfen, wie gut die Resultate sind, wenn weitere Aufnahmen gemacht würden, bei denen Spieler dieselben Trikots tragen. Sind die Resultate immer noch gut, könnte das Training einmal erfolgen und das Model anschliessend wiederverwendet werden. Dies ist aber nur bedingt praktisch, da immer unterschiedliche Spieler mitspielen und die Trikots nicht immer wieder verwendet werden können. Ausserdem ist fraglich, worauf die Klassifizierung hauptsächlich beruht. Wenn zum Beispiel gelernt wurde, dass Personen mit einem weissen Ärmel immer der Klasse «Ishaqh» angehören, funktioniert das System nicht mehr, wenn Ishaqh einmal mit schwarzem Shirt unter dem Überziehleibchen spielt. Für den Einsatz in einem Echtzeitsystem ist aber eine einfache Alternative gefunden worden: Ein weiteres Neuronales Netz, welches Zahlen erkennt. Ein solches sollte auf ähnliche Weise wie das hier gezeigte erstellt werden können. Dieses Netz müsste nicht jedes Mal vorgängig trainiert werden. Es müsste für ein Spielerbild erkennen, ob überhaupt eine Zahl sichtbar ist und wenn ja, welche. Somit bliebe nur noch die Frage, wie lange eine Klassifikation eines Bildes dauert. Das ist im Verlaufe dieser Arbeit nicht definitiv zu beantworten, aber essentiell für die Funktionsweise einer Echtzeitapplikation.

Sollte das Tracking in einer Nachverarbeitung geschehen, ist die Lösung annehmbar. Der Operator müsste für die ersten Minuten des Spieles das Tracking selbst ausführen. Das kann benutzerfreundlich umgesetzt werden und führt dazu, dass viele klassifizierte Bilder vorhanden sind. Anschliessend kann mit diesen in der gezeigten Weise ein Neuronales Netz trainiert werden, welches zur automatischen Fehlererkennung und Fehlerkorrektur dient. Danach wird der Operator nicht mehr benötigt.

## 6.5 Positionsbestimmung

Die Positionsbestimmung ist das Herzstück der Applikation. Die Lösung ist einfach zu berechnen und erfordert keine grossen Aufwände für die Konfiguration. Die Kameras können kalibriert werden, müssen aber nicht. Als Input dient wie bei der Farb- und Nummernerkennung ein Spielerbild, beziehungsweise die Position des Spielerbildes im Kamerabild. In der nachfolgenden Abbildung ist sichtbar, dass für

einen detektierten Spieler der unterste Punkt in der Mitte der Bounding Box für die Positions berechnung verwendet wird.



Abbildung 15 Der rote Punkt (in Pixelkoordinaten) wird für die Positionsumrechnung verwendet.

Die Positions berechnung basiert auf zwei unterschiedlichen Koordinatensystemen: Dem Perspektiven- und dem Modellsystem. Das Perspektivensystem bezieht sich auf die Sicht einer Kamera. Die Punkte können in Pixeldaten angegeben werden. Der Punkt [0, 0] befindet sich oben links im Bild und der Punkt [1920, 1080] ist bei einem Full-HD-Bild unten rechts zu finden. Der rote Standpunkt des Spielers auf der letzten Abbildung ist mit seinen Pixelkoordinaten im Perspektivensystem definiert. Jede Kamera hat ihr eigenes Perspektivensystem.

Das Modellsystem wiederum ist eine normalisierte Sicht auf das Spielfeld. Konkret ist es ein massstabsgereuer Plan des Spielfeldes, in dem 100 Pixel einem Meter entsprechen. Dieser Plan wurde durch Ausmessen der Halle erstellt. Er entspricht deren Abbild und dient dem Darstellen der Tracking-Ergebnisse.

### 6.5.1 Perspektiven- und Modellsystem

Das Modellsystem besteht aus einer Menge von Referenzpunkten, deren Positionen in der Halle genau bekannt sind. Dadurch entsteht ein Hallenplan in Modellkoordinaten. Jeder Referenzpunkt hat eine Identifikationsnummer, welche sich nicht ändert. Diese Punkte werden durch ein einmaliges Ausmessen des Spielfeldes ermittelt. Jeder Referenzpunkt ist im Code einprogrammiert und entsprechend konfigurierbar. Eine Konfiguration besteht aus den Positionen der Referenzpunkte im Modell und im jeweiligen kameraspezifischen Bild. Wie das genau funktioniert, ist im Benutzerhandbuch beschrieben.

In der nachfolgenden Grafik sind alle von einer Kamera sichtbaren Referenzpunkte eingezeichnet. Die nicht sichtbaren Punkte sind in der Applikation trotzdem konfiguriert und werden von anderen Kameras verwendet.

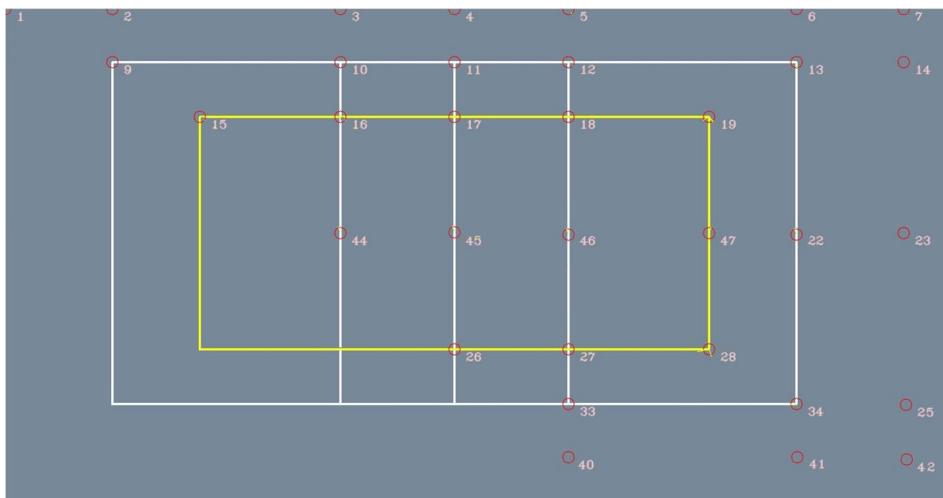


Abbildung 16 Plan der im Projekt verwendeten Halle. Er wurde massstabsgereu in Modellkoordinaten erfasst. Auch sichtbar sind einige Referenzpunkte.

Das Perspektivensystem beschreibt die Sicht, welche eine Kamera auf das Spielfeld hat. Pro Kamera existiert ein eigenes Perspektivensystem. Im untenstehenden Beispiel sind die für diese Kamera gefundenen Referenzpunkte durch rote Kreise dargestellt. In gewisser Weise wird die Wirklichkeit in ein Modell umgewandelt. Die in Abbildung 16 sichtbaren Referenzpunkte (im Modellsystem) sind auch in

der untenstehenden Abbildung (im Perspektivensystem der Kamera) sichtbar und anhand der ID erkennbar. Für eine Kamera werden die Informationen der Referenzpunktpositionen im Modell- sowie Perspektivensystem als Konfiguration bezeichnet.

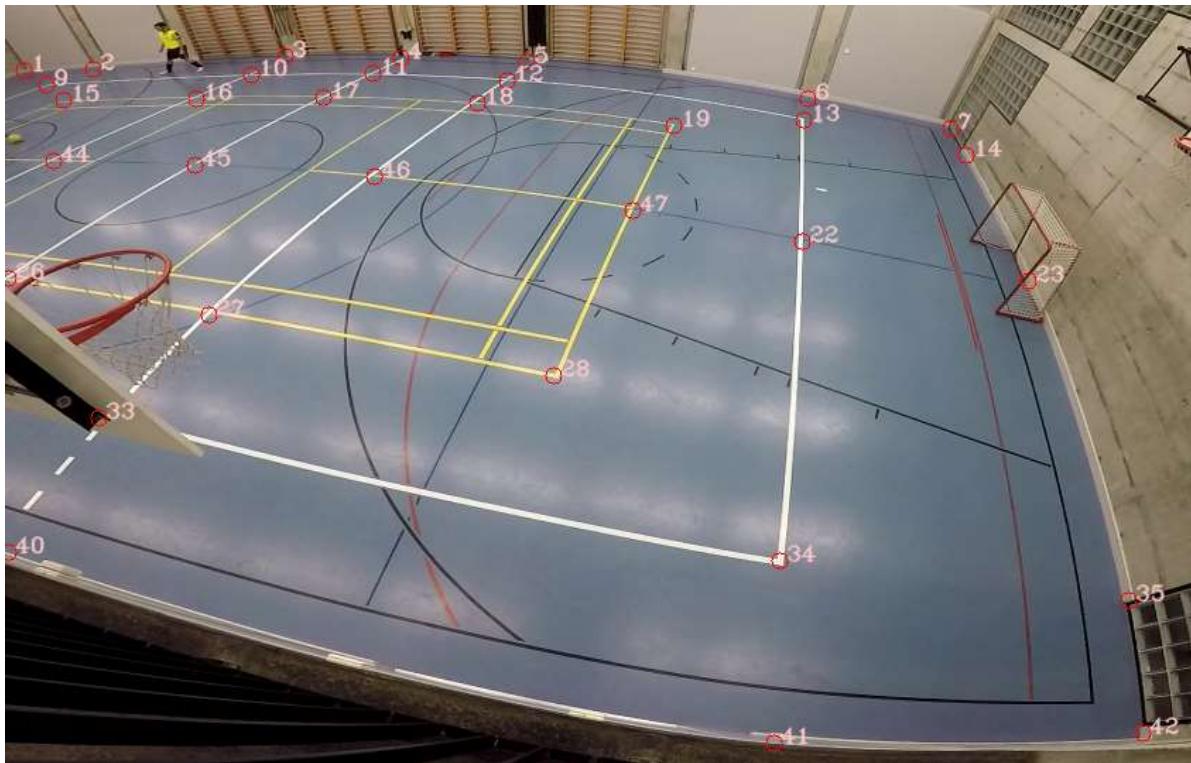


Abbildung 17 Referenzpunkte in einem Perspektiven-Kamera-Bild.

Sobald die Kamera ihre Sicht auf die Dinge verschiebt oder selbst die Position ändert, müssen die Referenzpunkte erneut gefunden werden. Da diese meist in bestimmten Eckpunkten zu finden sind, ist denkbar, auch sich bewegende Kameras zu integrieren. Dies könnte ermöglicht werden, indem die Punkte automatisch bestimmt werden. Das war jedoch außerhalb der Anforderungen dieses Projektes und könnte zu einem späteren Zeitpunkt eingebaut werden.

Die folgenden Unterkapitel behandeln nun die auf den beiden Systemen basierende Berechnung der Modellkoordinaten aus einem Perspektivenpunkt.

### 6.5.2 Grundsätzliches Konzept

Die von YOLO extrahierten Spielerbilder haben eine Position im Perspektivensystem der jeweiligen Kamera. Das ist in Abbildung 18 ersichtlich. Die Frage ist, welchen Modellkoordinaten diese Position entspricht. Dazu werden mit Hilfe der Konfiguration Berechnungen vollzogen. Als Konfiguration wird, wie erwähnt, eine Liste von Modell- und Perspektivenkoordinaten der Referenzpunkte pro Kamera bezeichnet. Es müssen für die Berechnung drei Referenzpunkte ausgewählt werden. Auf welche zwei Arten die Referenzpunkte ausgewählt werden, ist in den folgenden zwei Kapiteln beschrieben. Wurden die drei Referenzpunkte ermittelt, wird die Spielerposition anhand dieser in der Perspektive mit baryzentrischen Koordinaten ausgedrückt. Daraus resultieren die drei Werte  $u$ ,  $v$  und  $w$ , welche die Anteile von den in den nebenstehenden Abbildungen sichtbaren Punkten 8, 9 und 6 an der Spielerposition darstellen. Dieser Vorgang wird durch Abbildung 19 dargestellt. Mit diesen drei Werten wird mit den Modellkoordinaten der drei gewählten Referenzpunkte die Spielerposition im Modell berechnet. Ermöglicht wird das durch den Fakt, dass die Position der Referenzpunkte sowohl im Modellsystem wie auch im aktuellen Kamera-Perspektivenbild bekannt ist. Das wird in Abbildung 20 dargestellt.

Es ist wichtig zu erwähnen, dass die Implementation nicht auf das Vorhandensein aller Referenzpunkte angewiesen ist. So kann nur ein Teilbereich des Spielfeldes sichtbar sein und die Funktionsweise ist trotzdem sichergestellt.

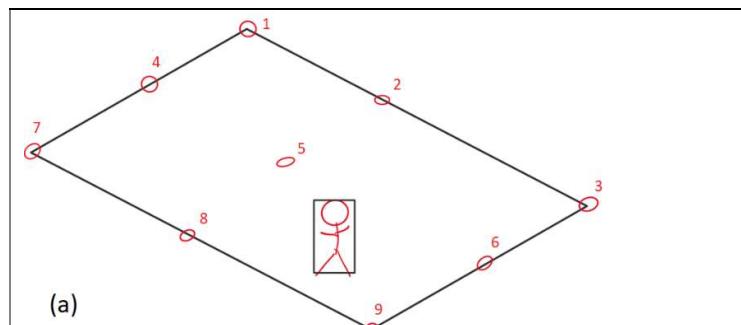


Abbildung 18 Ein Spieler steht im Perspektivensystem einer Kamera. Es müssen drei Referenzpunkte ausgewählt werden.

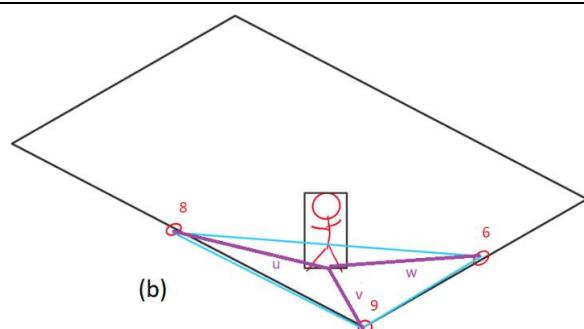


Abbildung 19 Es wurden drei Referenzpunkte für die Spielerposition ausgewählt. Die Position wird nun anhand dieser Referenzpunkte in baryzentrischen Koordinaten ( $u$ ,  $v$ ,  $w$ ) ausgedrückt.



Abbildung 20 Die baryzentrischen Koordinaten ( $u$ ,  $v$  und  $w$ ) aus der letzten Abbildung werden im Modellsystem verwendet, um die Spielerposition anzunähern.

Wie diese Umwandlung für mehrere Spieler aussieht, wird in den Abbildungen in der Folge gezeigt. Die grünen Umrandungen in Abbildung 21 stellen die von YOLO erkannten Spielerbereiche dar. Auch die Referenzpunkte werden angezeigt, wenn auch etwas schlechter sichtbar. Jeder dieser Spielerpunkte wurde auf die genannte Weise aus dem Perspektivensystem ins Modellsystem übertragen. Das Resultat dieser Transformation ist in der Abbildung 22 ersichtlich. Die roten Punkte sind die Referenzpunkte, die schwarzen Punkte sind die Spieler. Die gelben Linien, welche von den Spielerpunkten zu den Referenzpunkten führen, zeigen, welche Referenzpunkte für die baryzentrische Berechnung verwendet wurden.



Abbildung 21 Ein Ausschnitt eines Perspektivbildes einer Kamera mit den darin erkannten Spielern.

Die Zahlen, welche in der obigen Abbildung bei den Spielern stehen, können verwendet werden, um den Spieler im unten abgebildeten Modellsystem wiederzufinden.

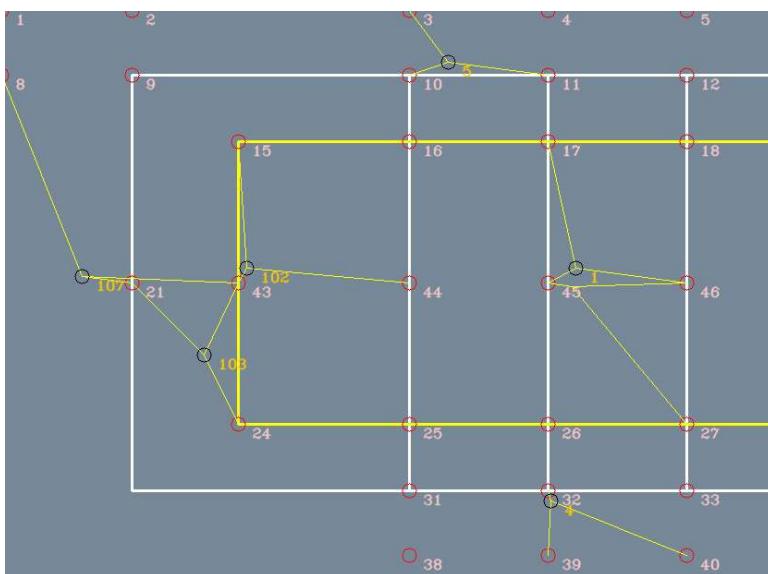


Abbildung 22 Die errechneten Positionen der Spieler in der letzten Grafik, dargestellt im Modellsystem inkl. der zur Berechnung verwendeten Referenzpunkte (gelbe Linien).

### 6.5.3 Referenzpunktauswahl – Dreieckslösung

Dies ist die erste Methode, um die drei Referenzpunkte für die Positionsbestimmung auszuwählen. Um diese Lösung umzusetzen, muss das Feld einmalig in Dreiecke unterteilt werden. Das geschieht, indem jeweils drei Referenzpunkte zusammen gespeichert werden. Konkret wird dies direkt im Code implementiert. Das ergibt ein sich über das gesamte Feld ziehendes Netz von Dreiecken.

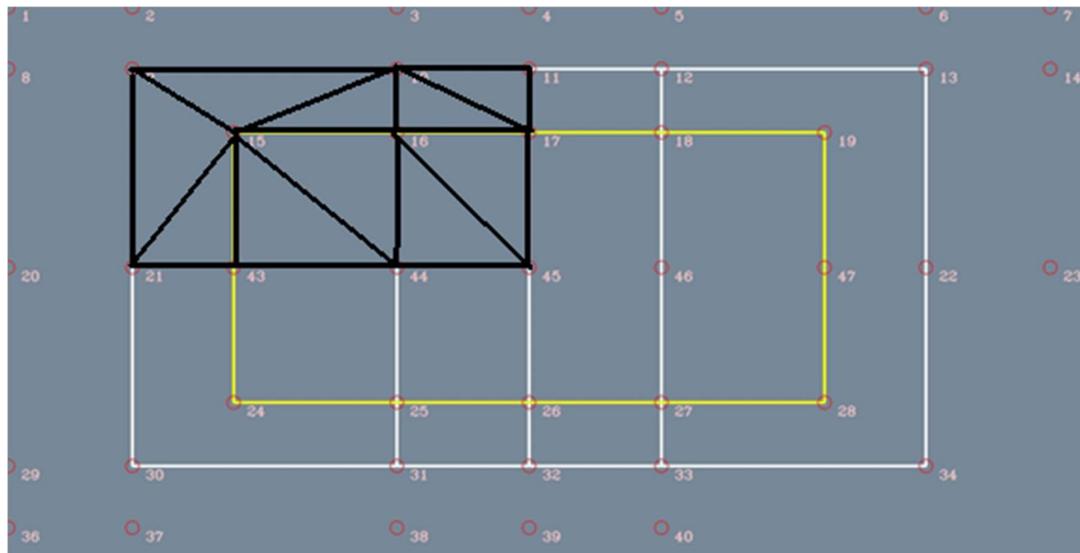


Abbildung 23 Ein Teil der Dreiecke ist hier in schwarz dargestellt. Im vollständig konfigurierten Programm ist das ganze Feld in Dreiecke aufgeteilt.

Um die Position zu bestimmen, wird nun für jedes konfigurierte Dreieck geprüft, ob die benötigten Referenzpunkte auf dieser Kamera sichtbar sind. Wenn nein, wird das Dreieck verworfen und das nächste ausprobiert. Wenn doch, kann dieses Dreieck geprüft werden.

Bei der Prüfung wird ermittelt, ob der Spielerpunkt (im Perspektivensystem) innerhalb einem der definierten Dreiecke liegt. Das könnte mit einer baryzentrischen Rechnung gelingen. Umgesetzt wurde jedoch eine andere Lösung. Damit wird bestimmt, ob der Spielerpunkt immer auf der gleichen Seite aller Dreiecksseiten liegt.

Die Ecken des Dreiecks werden A, B und C genannt.

Das Vorzeichen für eine Dreiecksseite mit den Punkten A und B wird wie folgt berechnet [22]:

$$\text{vorzeichenAB} = (\text{Spielerpunkt.x} - \text{B.x}) * (\text{A.y} - \text{B.y}) - (\text{A.x} - \text{B.x}) * (\text{Spielerpunkt.y} - \text{B.y})$$

Die Berechnung wird auch für die anderen Seiten ausgeführt:

$$\text{vorzeichenBC} = (\text{Spielerpunkt.x} - \text{C.x}) * (\text{B.y} - \text{C.y}) - (\text{B.x} - \text{C.x}) * (\text{Spielerpunkt.y} - \text{C.y})$$

$$\text{vorzeichenCA} = (\text{Spielerpunkt.x} - \text{A.x}) * (\text{C.y} - \text{A.y}) - (\text{C.x} - \text{A.x}) * (\text{Spielerpunkt.y} - \text{A.y})$$

Der Spielerpunkt befindet sich nun im Dreieck, wenn die Vorzeichen entweder alle positiv oder alle negativ sind.

Wenn der Punkt innerhalb des Dreieckes liegt, können die drei Punkte, welche das Dreieck bilden, für die folgenden Berechnungen verwendet werden. Andernfalls wird in der Iteration fortgefahrene, bis alle Dreiecke geprüft wurden.

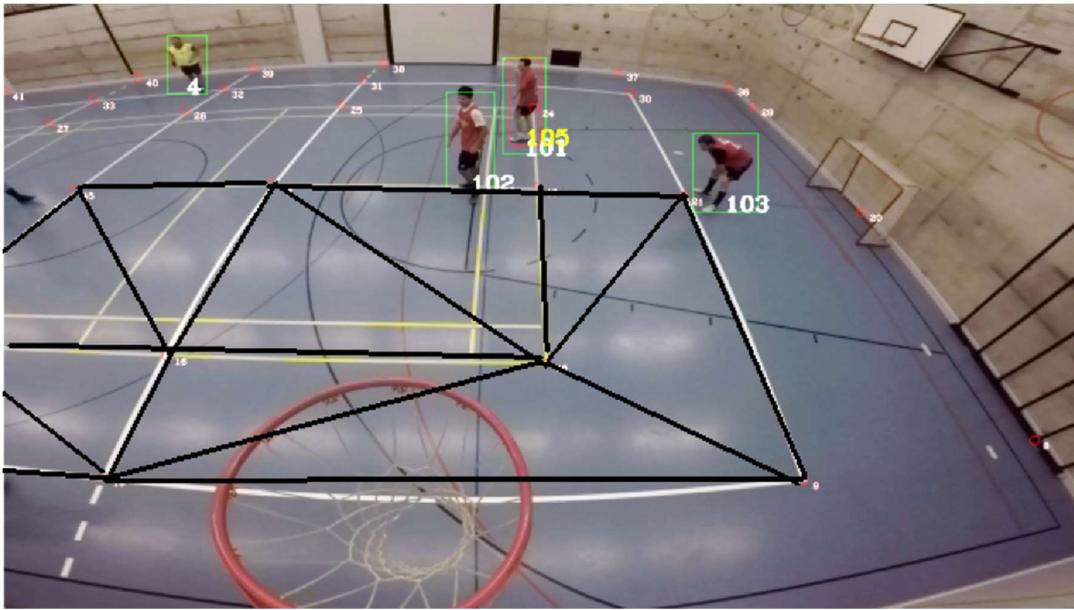


Abbildung 24 Die gleichen Dreiecke der letzten Abbildung nun in der Perspektive.

Wie aus der vorhergehenden Abbildung hervor geht, liegen die Linien der Dreiecke nicht immer exakt auf den Spielfeldlinien. Das führt zu einer minimalen Ungenauigkeit bei der baryzentrischen Annäherung. Diese Abweichungen fallen jedoch aufgrund der kleinen kritischen Gebiete nicht merkbar ins Gewicht.

#### 6.5.4 Referenzpunktauswahl – Sonderfalllösung

Wird mit der Dreiecklösung kein Dreieck gefunden, welches den Spieler beinhaltet, wird mit dieser Alternativlösung fortgefahrene. Hier wird über alle für die aktuelle Kamera verfügbaren Referenzpunkte iteriert und die drei Punkte gesucht, welche dem Spielerpunkt im Perspektivensystem am nächsten sind. Verglichen wird dabei die euklidische Distanz in Pixel. Dazu in der untenstehenden Abbildung ein Beispiel.

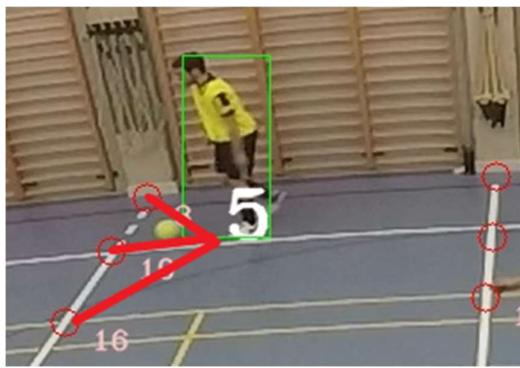


Abbildung 25 Die dem Spieler "5" nächsten Referenzpunkte wurden rot mit ihm verbunden.

Es gibt hier noch Ausschlusskriterien, welche verhindern, dass ein Punkt gewählt wird. So darf ein neu gewählter Punkt nicht in einer Linie mit den anderen beiden gewählten Punkten sein. Dies wird durch eine rein mathematische Prüfung sichergestellt und verhindert beispielsweise eine Auswahl wie in der vorgehenden Abbildung. Konkret wird geprüft, dass die drei Punkte nicht dieselben x- oder y-Modell-Koordinaten haben. Da nicht alle ungünstigen Kombinationen von Referenzpunkten in einer exakten Linie liegen, besteht die Möglichkeit, eine Kombination von drei Referenzpunkten explizit in der Konfiguration im Code auszuschliessen, indem man die drei Identifikatoren in einem entsprechenden Vektor speichert. Wie das genau funktioniert, ist in der Benutzerdokumentation erklärt.

### 6.5.5 Baryzentrische Berechnung

Anhand der in den letzten beiden Kapiteln erhaltenen drei Referenzpunkte kann nun mit einer baryzentrischen Rechnung die Position des Spielers im Modellsystem angenähert werden.

Dabei wird der Spielerpunkt in Bezug auf das von den drei Referenzpunkten gebildete Dreieck ausgedrückt. Die drei Referenzpunkte werden im Perspektivensystem mit einer der beiden gerade beschriebenen Methoden ausgewählt. Die Position der Referenzpunkte im Modellsystem ist aufgrund der konfigurierbaren Referenzpunkte pro Kamera auch bekannt und kann nun für die Berechnung verwendet werden. Für die in einer Kamera sichtbaren Referenzpunkte kennt der Algorithmus also die Position im Kamerabild (Perspektivensystem) und in der Halle (Modellsystem). Der untenstehenden Berechnung werden die Punkte in Perspektivenkoordinaten übergeben.

Konkret werden die drei Referenzpunkte als A, B und C bezeichnet. Punkt P soll in baryzentrischen Koordinaten ausgedrückt werden. Dabei kann die Position von P als Ebene ausgedrückt werden [13]:

$$P = A + v(B - A) + w(C - A)$$

v und w müssen bestimmt werden. u ist eigentlich redundant und kann aus v und w berechnet werden:

$$u = 1 - v - w$$

Die Ebenengleichung wird etwas umgestellt und die Subtraktionen durch Variablen ersetzt:

$$P - A = v(B - A) + w(C - A)$$

$$v0 = B - A$$

$$v1 = C - A$$

$$v2 = P - A$$

$$v * v0 + w * v1 = v2$$

Nun kann ein 2x2 System von Gleichungen erstellt werden, indem auf beiden Seiten das Skalarprodukt mit v0 und anschliessend v1 gebildet wird:

$$(v * v0 + w * v1) \circ v0 = v2 \circ v0$$

$$(v * v0 + w * v1) \circ v1 = v2 \circ v1$$

Da das Skalarprodukt linear ist, kann das umgestellt werden:

$$v(v0 \circ v0) + w(v1 \circ v0) = v2 \circ v0$$

$$v(v0 \circ v1) + w(v1 \circ v1) = v2 \circ v1$$

Eine Vereinfachung erfolgt:

$$d00 = v0 \circ v0$$

$$d01 = v0 \circ v1$$

$$d10 = v1 \circ v0$$

$$d11 = v1 \circ v1$$

$$d20 = v2 \circ v0$$

$$d21 = v2 \circ v1$$

Das ergibt:

$$v d00 + w d10 = d20$$

$$v d01 + w d11 = d21$$

Die Skalarprodukte können berechnet werden. Anschliessend werden die Unbekannten mit Hilfe der Cramerschen Regel bestimmt. Im Code der Applikation wird d10 aus Effizienzgründen nie berechnet, sondern d01 verwendet. Das wird möglich, weil das Skalarprodukt symmetrisch ist.

$$u = 1 - v - w \quad v = \frac{\det(d20 \quad d10)}{\det(d00 \quad d10)} \quad w = \frac{\det(d00 \quad d20)}{\det(d01 \quad d21)}$$

u, v und w können nun im Modellsystem verwendet werden, um die Annäherung der Position zu bestimmen. Mithilfe der Modellkoordinaten der Referenzpunkte A, B und C wird nun die angenäherte Position des Spielers («Modellposition») berechnet.

$$\text{Modellposition} = u * A + v * B + w * C$$

## 6.6 Tracking-Modul

Das Tracking-Modul bekommt als Input alle von den Kameras erfassten Spieler. Konkret sind das drei Listen mit Spielerobjekten, welche Modell-Position und Farbe enthalten. Die Koordinaten der Spielerposition sind bereits im Modellsystem und dementsprechend vergleichbar. Spieler, die von mehreren Kameras gesehen werden, erscheinen auch mehrmals im Input.

Die Aufgabe des Tracking-Moduls ist es, die zu trackenden Spieler einem Spielerobjekt aus dem Input zuzuordnen.

In der Abbildung 26 ist ersichtlich, wie die Situation von den Kameras wahrgenommen wird. Zudem sind in folgender Liste alle Spieler und ihre Sichtbarkeit definiert.

Spieler	Kamera, Positionierung im Bild bezüglich Teamkollegen (Nummernreferenz)
Gelb, #2	Kamera 1, unterer Spieler (1001) Kamera 2, rechter Spieler (2001) Kamera 3, linker Spieler (3002)
Gelb, #8	Kamera 1, oberster Spieler (1002) Kamera 2, oberster Spieler (2004) Kamera 3, von YOLO nicht erkannt
Gelb, #9	Kamera 1, nicht sichtbar Kamera 2, unterster Spieler (2003) Kamera 3, oberster Spieler (3005)
Rot, #2	Kamera 1, nicht sichtbar Kamera 2, unterster Spieler (2102) Kamera 3, oberster Spieler (3101)
Rot, #8	Kamera 1, nicht sichtbar Kamera 2, von YOLO nicht erkannt Kamera 3, rechter Spieler (3103)
Rot, #9	Kamera 1, nicht sichtbar Kamera 2, oberster Spieler (2105) Kamera 3, unterster Spieler (3104)

Tabelle 8 Spielerindex für nebenstehende Abbildung.

Die in der obigen Tabelle ersichtbare «Nummernreferenz» kann verwendet werden, um den Spielerpunkt in der Abbildung 27 wiederzufinden. In ebenjener Abbildung ist auch als Gesamtbild der Input für das Tracking-Modul sichtbar, welcher aus den drei Kamerasichten (Abbildung 26) entstanden ist. Es ist, wie es für eine übersichtliche Situation als üblich angesehen werden kann, eine klare Gruppierung der Spielerpunkte sichtbar. Es gibt einige Ungenauigkeiten, welche aus minimalen Verschiebungen der Bounding Box oder aus unpräzisen Positions berechnungen stammen. Trotzdem ist es möglich, das Bild, welches ein Mensch sich aus den Kamerasichten macht, in der Gesamtsicht auf dem Modell wiederzuerkennen.

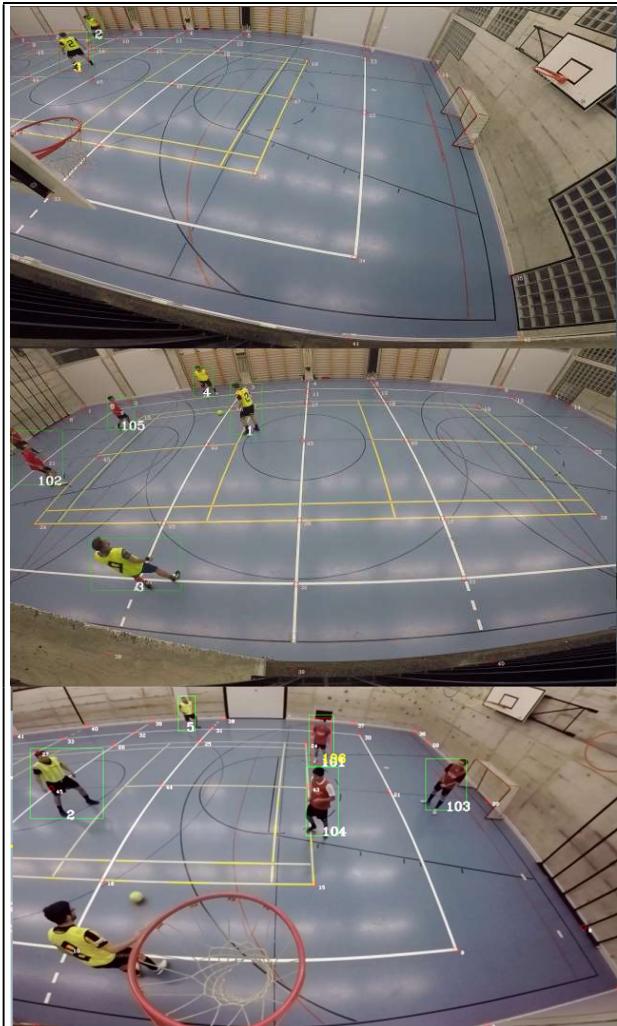


Abbildung 26 Die drei Kamerasichten auf das Geschehen zu einem zufälligen Zeitpunkt.

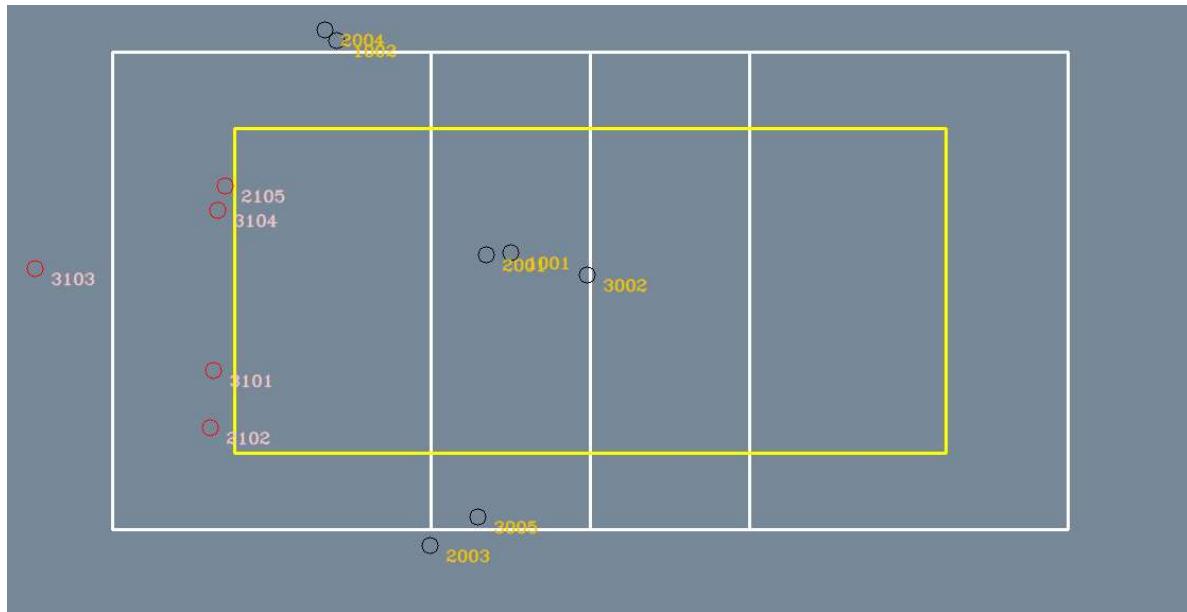


Abbildung 27 Der aus den in der letzten Abbildung gezeigten Kamerasichten entstandene Input für das Tracking-Modul.

### **6.6.1 Initiales Erkennen der Spieler**

Das initiale Erkennen der Spieler ist ein noch nicht vertieft angegangenes Problem. Initial werden einfach alle Spieler, die von der zentral positionierten Kamera «cameraHud» erkannt wurden, als zu trackende Spieler übernommen.

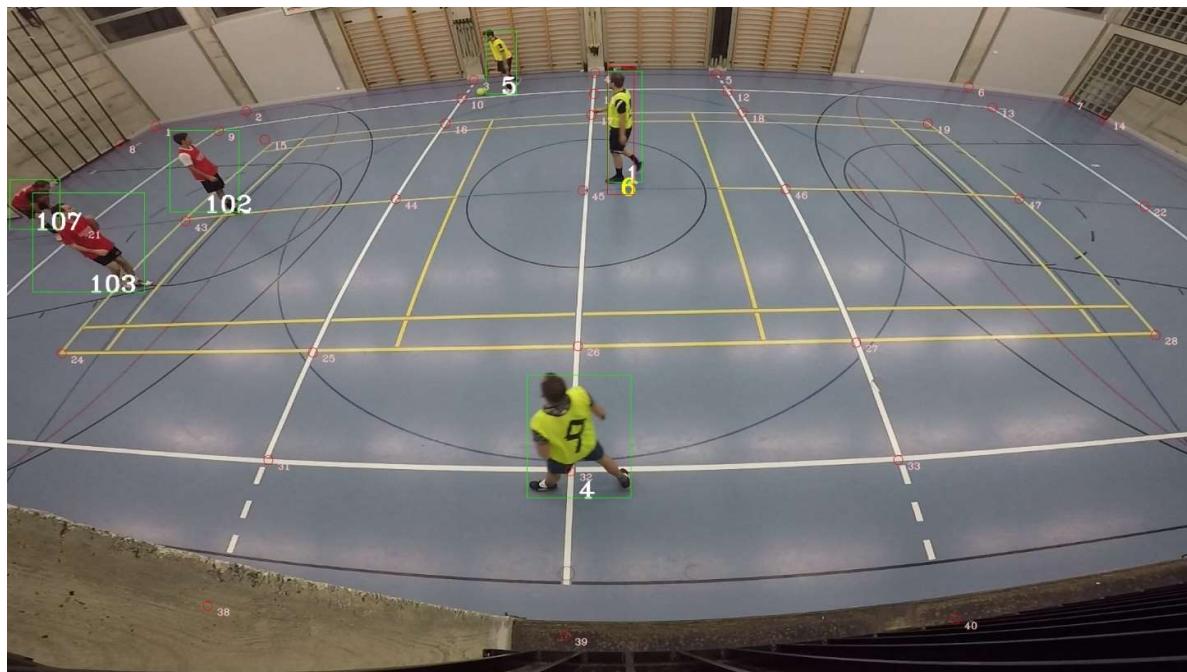


Abbildung 28 Das erste Frame der Kamera "cameraHud" im Testsegment.

Das setzt voraus, dass alle Spieler für diese Kamera sichtbar sind und nicht nahe beieinanderstehen. In der Testsequenz ist das gegeben. Für ein richtiges Produkt müsste eine bessere Lösung erarbeitet oder eine Anforderung an die Videodaten erfasst werden, die das sicherstellt.

### 6.6.2 Wiederfinden der bestehenden Spieler

Es wird versucht, die getrackten Spieler im Input des aktuellen Durchlaufs wiederzuerkennen. Das passiert, indem für jeden Spieler über alle Input-Spieler iteriert wird und der am nächsten liegende (innerhalb eines maximalen Radius) und farblich passende Spieler eruiert wird. Die Position dieses Input-Spielers wird nun als die neue Position des Spielers angenommen. Der Input-Spieler steht anschliessend nicht mehr zur Auswahl.

Wird ein getrackter Spieler im Input nicht wiedererkannt, wird sein Zuletzt-Gesehen-Zähler um eins erhöht und die bisherige Position weiterhin verwendet.

Als konkretes Beispiel wird in der Abbildung 29 exemplarisch die Suche nach einem bisher erkannten roten Spieler dargestellt. Der rote Stern stellt die im Tracking eruierte letzte bekannte Position dar. Spielerpunkte von als Rot erkannten Spielern werden rot dargestellt und die anderen schwarz. Drei Input-Punkte kommen nun als neue Position des Spielers in Frage. Mögliche andere Punkte (nicht sichtbar in der Grafik) können aufgrund einer Überschreitung der Maximaldistanz (grüner Kreis) ausgeschlossen werden. Da nun die Distanz zum Punkt 2105 kleiner ist, als diejenige zum Punkt 3104, wird angenommen, dass sich der Spieler neu am Punkt 2105 aufhält. Punkt 2003 kann, obwohl er am nächsten wäre, ausgeschlossen werden, da der Punkt nicht als Rot klassifiziert wurde und dementsprechend keinem roten Spieler zugeordnet wird.

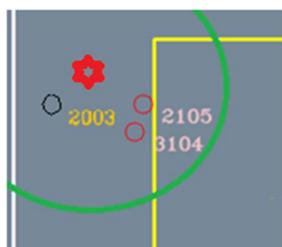


Abbildung 29 Beispieldiagramm zur Spielerwiederfindung.

Der maximal akzeptierte Radius beim Suchen nach einem passenden Spieler ist grundsätzlich 2m. Dazu addiert werden 70cm für jedes Frame, in dem der Spieler nicht gesehen wurde. Durch diese Ausweitung des Radius ist es möglich, verlorene Spieler wiederzufinden. Diese Ausweitung des Suchradius bringt aber auch das Risiko, Spieler zu verwechseln. Mit guten Positions berechnungsergebnissen sinkt die Möglichkeit, einen Spieler überhaupt zu verlieren.

Nebst der Methode, den nächsten gleichfarbigen Nachbarn auszuwählen, wurden auch andere Vorgehensweisen in Betracht gezogen. Zum Beispiel sollten gleichfarbige Spielerpunkte, wenn sie nahe genug beieinander liegen, zu einem vereint werden (Position durch Mittelung setzen). Das führt aber zu mehr Problemen, als gelöst werden. Wenn zum Beispiel zwei Spieler des gleichen Teams effektiv nahe beieinanderstehen und jeder nur von einer Kamera gefunden wird, kann es sein, dass der eine Spieler verloren geht.

Diese Idee ist aber erweiterbar: Was, wenn noch beachtet wird, wo die getrackten Spieler im letzten Frame standen und, wenn zwei nahe beieinanderstehen, eine Ausnahme hinzugefügt wird? In dem Fall könnten die zwei Punkte einzeln weiterverwendet werden.

Das Problem an diesem Punkt war während der Entwicklung, dass es sehr viele Möglichkeiten gab. Zudem führten viele in die Irre und zu weiteren Problemen und schlechteren Ergebnissen. Trotzdem kann gesagt werden, dass mit einer Verfeinerung dieses Punktes möglicherweise noch an Präzision dazugewonnen werden kann. Im Endeffekt war jedoch zu wenig Zeit vorhanden, um diesen Punkt noch mehr zu vertiefen. Zudem konnten die Fehlerfälle in der Testsequenz nicht auf einen Fehler beim Wiederfinden der Spieler zurückgeführt werden, sondern es lagen unpräzise Spielerdetektionen zugrunde.

### 6.6.3 Bevorzugung aktueller Spieler

Es wird für jeden Spieler, der getrackt wird, ein Zuletzt-Gesehen-Zähler geführt. Der Wert des Zählers entspricht der Anzahl Frames, seitdem der dazugehörige Spieler das letzte Mal erkannt wurde. Das ist nützlich und wird verwendet, um diejenigen Spieler, welche aktuell gut getrackt werden, bevorzugt zu behandeln. Konkret werden zuerst diejenigen Spieler, welche im letzten oder vorletzten Frame erkannt

wurden, behandelt. Diese Spieler werden vor den restlichen Spielern behandelt und es wird zuerst versucht, ihnen einen Punkt aus dem neuen Input zuzuweisen. So soll verhindert werden, dass verlorene Spieler die Inputpunkte von aktuell gut verfolgten Spielern verwenden.

## 6.7 Korrekturfunktion

Um einem Operator der Applikation die Möglichkeit zu eröffnen, fehlerhaft getrackte Spieler zu korrigieren, besteht eine Korrekturfunktion. In einem zu wählenden Zeitabschnitt werden alle Kamerabilder und das aktuelle Tracking-Resultat angezeigt. Der Operator kann nun beurteilen, ob das Resultat der Realität nahe genug ist. Bei Fehlerfällen kann er bis zu drei Mal in das Bild mit den Tracking-Resultaten klicken und diesen Positionen per Tastendruck neue Spieler zuordnen. Das übersteuert sofort das Tracking und führt dazu, dass die Spielerpositionen gemäss der Korrektur angepasst werden.

Für die aktuellen Anforderungen ist das genügend. Sobald die Performance aber eine Echtzeitverarbeitung erlaubt, sollte eine intuitivere und schnellere Methode entwickelt werden.

Zusätzlich kann erwähnt werden, dass die Korrekturfunktion im zukünftigen Zielbild höchstens noch ergänzend benötigt wird. Mithilfe der automatischen Fehlerkorrektur sollten die Ergebnisse automatisch besser sein.

## 6.8 Weitere Funktionen

Es wurden im Verlaufe des Projekts einige weitere Funktionen eingebaut.

### 6.8.1 Berechnung der Laufweglänge

Als Beispiel für eine statistische Berechnung werden für das Demonstrationsvideo die gelauftenen Distanzen der einzelnen Spieler berechnet. Die Berechnung ist einfach: Die euklidische Distanz von der letzten bekannten Position eines Spielers zu der neuen Position. Diese Distanzen können für einen Spieler aufsummiert werden. Da sich auch ein Spieler, welcher sich in der Realität nicht bewegt, aufgrund der leicht veränderten Personenerkennungs- und Positionsberechnungen leicht verschiebt und um andere Ungenauigkeiten auszugleichen, werden die summierten Distanzen noch mit einem Faktor multipliziert, welcher sie etwas verkleinert. Als Wert des Faktors wurde 0,8 gewählt. Es wurde nicht geprüft, wie genau die Distanzberechnung ist. Sie verhält sich jedoch wie erwartet: Wenn ein Spieler rennt, beschleunigt sich der Distanzzähler. Wenn der Spieler stehen bleibt, bleibt auch der Zähler konstant.

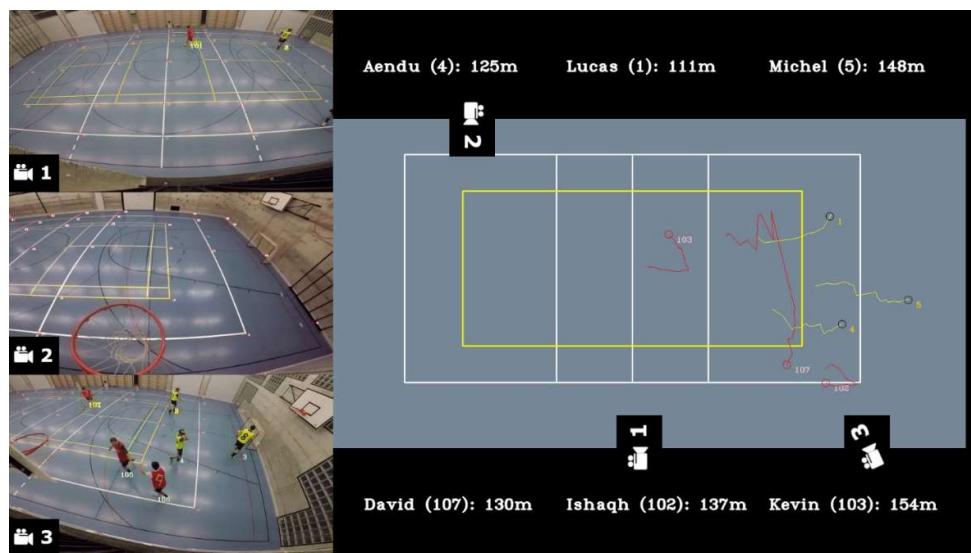


Abbildung 30 Ein Frame des Demonstrationsvideos gegen Ende der bearbeiteten Sequenz.

In der obenstehenden Abbildung sind die berechneten Laufdistanzen aufgeführt. Eine stichhaltige Prüfung der errechneten Distanzen wurde aus zeitlichen Gründen nicht vorgenommen. Diese Auswertung soll den Nutzen der Applikation für statistische Auswertungen aufzeigen. Wie das Demonstrationsvideo entsteht, wird in einem späteren Kapitel beschrieben.

### **6.8.2 Logging**

Es besteht eine eigene Logger-Klasse, welche es für jeden Log-Aufruf erlaubt, ein Log-Level mitzugeben. Ist dieses kleiner als 1, wird die dazugehörige Nachricht nur angezeigt, wenn der Debug-Modus aktiv ist. Dieser Modus kann direkt auf der Logger-Klasse gesetzt werden.

### **6.8.3 Debug-Informationen und Bilder**

Es werden diverse Debug-Informationen in der Form von Bildern erstellt und es besteht die Möglichkeit, diese auszugeben. Die ganze Auswahl an Bildern und die Bedeutung der enthaltenen Informationen wird im Benutzerhandbuch genauer erörtert.

### **6.8.4 Tracking-Video exportieren**

Es besteht die Möglichkeit, ein Video mit den Ergebnissen des Trackings zu erstellen und dieses auf der Festplatte abzuspeichern. Das passiert in der Methode `createVideo` der eigenen Klasse `TrackingModule`. Dafür werden die Tracking-Resultate als Bilder im Arbeitsspeicher behalten, sodass anschliessend daraus ein Video erstellt werden kann. Diese Möglichkeit wird nach dem Erstellen des Tools `VideoCreator` jedoch nicht mehr wirklich benötigt. Sobald die Applikation in Echtzeit funktioniert, kann der `VideoCreator` in diese Methode übertragen werden.

## **6.9 Tools**

Es wurden zur Vereinfachung der aufwändigen Arbeiten kleine Programme geschrieben. Die wichtigsten davon sind in der Folge kurz erklärt. Jedes Programm wird im Benutzerhandbuch noch vertieft behandelt und die Anwendung erörtert.

### **6.9.1 Groundtruth**

Es besteht eine kleine Applikation, welche es erlaubt, für drei Kamerawinkel den Standort eines Spielers zu bestimmen und auszugeben. Dafür können die Kamerawinkel angeschaut und die Position auf einem Modellbild des Spielfelds eingetragen werden. Diese Applikation vereinfacht und verschnellert die Erstellung der Groundtruth-Daten enorm.

### **6.9.2 Playerimageorganizer**

Um eine automatisierte Fehlerkorrektur zu erstellen, wurde ein Neuronales Netz in Betracht gezogen, welches Spieler anhand des Spielerbildes klassifizieren soll. Dafür wurden in der Testsequenz die Spielerbilder abgespeichert. Diese grosse Menge an Bildern musste nun klassifiziert werden, damit ein Trainieren des Netzes möglich wurde.

Diese Applikation zeigt ein Spielerbild an und ermöglicht es, dieses per Tastendruck in einen Unterordner zu verschieben. So können schnell und einfach Klassen gebildet werden.

### **6.9.3 Position**

Dieses Programm erlaubt es, ein Kamerabild anzuzeigen und darauf Punkte zu definieren. Dies ist sehr nützlich, wenn eine Kamerakonfiguration vorgenommen werden muss. So können die Referenzpunkte schnell durchgeklickt und die entsprechenden Koordinaten für die Konfiguration verwendet werden.

### **6.9.4 VideoCreator**

Aufgrund der mangelhaften Performance wurde beschlossen, die Demonstration auszulagern. Konkret funktioniert das so: Die richtige Tracking-Applikation wird mit der Testsequenz gestartet und verarbeitet die Daten wie gewohnt. Dabei werden jedoch zusätzlich zu den Bearbeitungen noch alle Kamerawinkel sowie die Tracking-Sicht als Bild mit sinnvollem Namen auf der Festplatte abgespeichert: `{frameId}_frameHud.jpg`, `{frameId}_frameMar.jpg`, `{frameId}_frameMic.jpg` und `{frameId}_tracking.jpg`.

So entstehen für die etwa 90 Sekunden dauernde Sequenz ungefähr 3500 Bilder. Zusätzlich wird pro Frame und Spieler die zwischen dem aktuellen und dem letzten Frame zurückgelegte Distanz ausgegeben. Dies passiert in der folgenden Form:

```
cout << "distances" << histPlayer.getCamerasPlayerId() << ".insert(std::make_pair(" << frameId << ", "<< distance <<"));" << endl;
```

Dadurch wird eine Code-Zeile generiert und in der Console ausgegeben, welche in der Initialisierungsmethode des VideoCreators eingefügt werden kann und eine dort vorhandene Map befüllt:

```
std::map<int, int> distances5;  
distances5.insert(std::make_pair(558, 18));
```

Diese Zeile bedeutet, dass der Spieler 5 (distances5) zwischen Frame 557 und 558 18 Pixel im Modellsystem zurückgelegt hat. Da hundert Pixel einem Meter entsprechen, entsprechen 18 Pixel 18 Zentimetern.

So werden die für das Video benötigten Informationen exportiert.

Der VideoMaker selbst iteriert über eine ID von 1 bis 882. Das entspricht aufgrund der 10 pro Sekunde verarbeiteten Frames den Indices aller Bilder während 88.2s. Mithilfe der ID können die korrekten Bilder und Distanzdaten ausgelesen werden. Alle ausgelesenen Informationen werden anschliessend zu einem Bild zusammengefügt und in einem Video gespeichert. Ein Beispieldframe ist in der Abbildung 30 zu sehen.

In dieser Weise wird das Performanceproblem für die Demonstration einfach und ohne die Daten zu verfälschen gelöst. Es ist nichts sichtbar, was im normalen Applikationsverlauf nicht auch sichtbar wäre.

## 6.10 Performance-Messungen

In diesem Unterkapitel wird die Aufteilung der Gesamlaufzeit in die verschiedenen Schritte aufgezeigt. So soll gezeigt werden, welche Arbeiten wie viel Prozent der Gesamtdauer ausmachen. Als Durchlauf wird in der Folge die Verarbeitung von drei Frames (eines pro Kamera) inklusive Tracking bezeichnet.

Als Beispieldaten dienen die Zeitmessungen eines Durchlaufs vom 24.12.2018 auf einer virtuellen Maschine (Ubuntu 16.04). Es wurden zufällig fünf Durchläufe ausgewählt und die Daten in die untenstehende Tabelle übertragen. Damit handelt es sich bei den Daten nicht um repräsentative Daten, doch es kann eine Tendenz für die Dauer der einzelnen Schritte abgeleitet werden. Die erfassten Daten (zumindest die prozentualen Verhältnisse) stimmen für alle durchgeföhrten Tests auf verschiedenen Betriebssystemen (Ubuntu 16.04, Windows, Ubuntu 18.04) und auf verschiedenen Maschinen (VM, Notebook, Gaming Computer) gut überein.

	Bilder auslesen	Personenerkennung	Spielerextraktion und Positionsbestimmung	Tracking Logik	Summe
<b>Frame 1 (s)</b>	0,19106	1,100873	0,036686	0,012511	1,34113
<b>Frame 1 (%)</b>	14,25%	82,09%	2,74%	0,93%	100%
<b>Frame 2 (s)</b>	0,096068	1,076111	0,037764	0,013171	1,223114
<b>Frame 2 (%)</b>	7,85%	87,98%	3,09%	1,08%	100%
<b>Frame 3 (s)</b>	0,097033	1,590043	0,040979	0,012512	1,740567
<b>Frame 3 (%)</b>	5,57%	91,35%	2,35%	0,72%	100%
<b>Frame 4 (s)</b>	0,099245	1,151196	0,03791	0,012911	1,301262
<b>Frame 4 (%)</b>	7,63%	88,47%	2,91%	0,99%	100%
<b>Frame 5 (s)</b>	0,167482	1,093667	0,037532	0,012543	1,311224
<b>Frame 5 (%)</b>	12,77%	83,41%	2,86%	0,96%	100%
<b>Mittelwert (s)</b>	0,1301776	1,202378	0,0381742	0,0127296	
<b>Mittelwert (%)</b>	9,62%	86,66%	2,79%	0,94%	

Tabelle 9 Performancemessung für fünf Durchläufe.

Es können folgende Schlüsse gezogen werden:

- Die Personenerkennung mit YOLO dauert etwa 1,2s und macht ungefähr 87% der Gesamtdauer des Programms aus.
- Die restlichen Arbeiten können in etwa 0,2s ausgeführt werden.

Offensichtlich stellt die Personenerkennung mit YOLO ein Problem dar. Das Ziel der Echtzeitverarbeitung ist damit auf die Schnelle nicht zu erreichen. Die restliche Verarbeitung ohne Personenerkennung würde eine Echtzeitverarbeitung mit mindestens 5FPS erlauben.

## 6.11 Performance-Verbesserungen YOLO

Es kann festgestellt werden, dass die Personenerkennung mit YOLO zu langsam für eine Echtzeitauswertung ist. Deswegen wurden diverse Verbesserungen und Beschleunigungen überprüft und ausprobiert. Es muss angemerkt werden, dass YOLO deutlich schneller ist, wenn es außerhalb von OpenCV direkt auf Darknet verwendet wird [31]. OpenCV ist aber das Framework der Wahl und ein Wechsel im Laufe des Projekts nicht denkbar. Die Geschwindigkeiten, welche in diesem Projekt erreicht wurden, sind ähnlich wie die von anderen Nutzern von YOLO via OpenCV [31].

### 6.11.1 Hardware

Die Hardware ist für die Entwicklung jeweils gegeben. Es wird ein Lenovo Yoga 720 verwendet, welches einen Prozessor von Intel (i7-7700HQ) und eine Grafikkarte von nVidia (GeForce GTX 1050) beinhaltet [14]. Bei den Durchläufen auf einer virtuellen Maschine, welche auf dem Lenovo Yoga laufen, können keine Geschwindigkeitseinbussen gegenüber den Durchläufen direkt auf der Maschine festgestellt werden. Ein Durchlauf dauert zwischen 1.2 und 1.8 Sekunden.

Performanceverbesserungen können auf einem vom CPVR-Lab zur Verfügung gestellten Gaming Computer (Intel i7-7700K und nVidia GeForce GTX 1080) festgestellt werden. So dauert die Verarbeitung eines kompletten Durchlaufes (Bilder auslesen, Personenerkennung, Spielerextraktion, Positionsbestimmung und Tracking-Logik) nur noch ungefähr 0,75s und kann dementsprechend deutlich verringert werden.

Die Berechnungen liefern bisher nur auf dem Prozessor. Es ist also denkbar, mit einem noch schnelleren Prozessor noch bessere Resultate zu erreichen.

In einem der folgenden Unterkapitel wird auch die Berechnung der Personenerkennung mit YOLO auf der Grafikkarte analysiert.

### 6.11.2 YOLO-Konfiguration

Es ist mit YOLO möglich, die Input-Bilder zu skalieren. In der Standardkonfiguration werden Bilder mit der Grösse 416px auf 416px verwendet. Eine andere mögliche Grösse wäre 224px auf 224px. Damit verkleinert sich die Dauer der Personenerkennung gemäss eigenen Tests um etwa 30%. Das Problem ist jedoch, dass die resultierende Detektion deutlich schlechter ist und dementsprechend zu schlechteren Ergebnissen führt.



Abbildung 31 Beispiele für die schlechtere Erkennung mit der kleineren Skalierung. Links: Doppelte Erkennungen. Rechts: Fehlende Erkennung.

Möglicherweise sind die schlechteren Detektionsergebnisse mit angepasster Programmlogik ausgleichbar. Während der Dauer dieses Projektes wurde aber davon abgesehen, da die Performanceverbesserung zu klein ist und der Aufwand für eine Anpassung der Programmlogik zu gross würde.

### 6.11.3 TinyYolo

Es gibt neben der normalen Version auch eine Minimalversion von YOLO. Diese nennt sich TinyYolo [8]. Die Performance kann wie bei der YOLO-Konfigurationsanpassung etwas verbessert werden, doch die Resultate genügen den Ansprüchen ebenfalls nicht. Dementsprechend wurde dieser Lösungsansatz schnell verworfen.

### 6.11.4 Anpassung von Target und Backend

In OpenCV können für ein Neuronales Netz verschiedene Backends und Targets verwendet werden. Als Backend wird die verwendete Berechnungsmethode bezeichnet. Mögliche Optionen sind DNN\_BACKEND\_OPENCV, DNN\_BACKEND\_INFERENCE\_ENGINE sowie DNN\_BACKEND\_HALIDE. Halide ist ein Opensource-Projekt, um Bildverarbeitungs-Algorithmen zu schreiben und die Ausführung auf verschiedenen Geräten (CPU, Grafikkarte) zu verteilen [23].

Das Target wiederum bezeichnet das ausführende Gerät. Ausgewählt werden kann zwischen DNN\_TARGET\_CPU, DNN\_TARGET\_OPENCL, DNN\_TARGET\_OPENCL\_FP16, DNN\_TARGET\_MYRIAD und DNN\_TARGET\_FPGA. Getestet wird ausschliesslich die Verwendung von DNN\_TARGET\_CPU und DNN\_TARGET\_OPENCL, welche das Ansteuern von CPU und Grafikkarte ermöglichen [15].

Auf der nVidia GeForce GTX 1080 des Gaming Computers wird versucht, das Berechnen der Personenerkennung auf der Grafikkarte laufen zu lassen. Zudem werden die Laufzeiten der verschiedenen Backends verglichen.

	DNN_BACKEND_OPENCV	DNN_BACKEND_HALIDE		
	OpenCV 3.4.4	OpenCV 3.4.2	OpenCV 3.4.4	OpenCV 3.4.2
DNN_TARGET_CPU	~0,75s	~0,75s	~1,75s	~1,75s
DNN_TARGET_OPENCL	~1,2s	~3,8s	Fehler	Fehler

Tabelle 10 Gesamtdauervergleich eines Durchlaufs anhand von Backend und Target.

Es erstaunt etwas, dass die schnellste Methode DNN\_BACKEND\_OPENCV auf der CPU ist. Dasselbe Backend auf der Grafikkarte ist deutlich langsamer, obwohl mit dem Kommandozeilentool nvidia-smi nachgewiesen werden konnte, dass eine der beiden Grafikkarten komplett ausgelastet war. Die eigenen Messungen bestätigen jedoch diejenigen der OpenCV-Entwickler, welche diese auf Github veröffentlicht haben [16]. Begründet wird dies in den meisten Foren damit, dass DNN\_BACKEND\_OPENCV stark optimierte Operationen verwendet und die OpenCV Entwickler empfehlen den Einsatz von Halide nicht [29].

### 6.11.5 OpenVino

Intel hat für ihre Plattform ein Toolkit veröffentlicht, welches die Performance speziell im Bereich der Neuronalen Netze scheinbar verbessern soll. Dieses stellt ein neues Backend in OPENCV dar: DNN\_BACKEND\_INFERENCE\_ENGINE [17]. Inference steht für die Verwendung eines bereits trainierten Neuronalen Netzes zur Klassifikation von neuen Daten [30].

Die Installation auf Windows 10 ist jedoch trotz mehreren Versuchen nicht gelungen. Zudem kann der Gaming Computer des CPVR-Labs nicht verwendet werden, da OpenVino nicht mit Ubuntu 18.04 kompatibel ist. In den entsprechenden Foren wird empfohlen, mit OpenVino zu arbeiten. Bei genauerem Vergleich ist OpenVino aber offenbar nur 1.03 mal schneller als das normale OpenCV-Backend [30]. Dieser Lösungsansatz konnte nicht selbst erarbeitet werden. Er scheint im Moment noch nicht alltagstauglich zu sein und es hätte zu viel Zeit in Anspruch genommen, diesen Ansatz zu vertiefen.

Die Intel Inference Engine ist seit kurzer Zeit (Dezember 2018) direkt im Github von OpenCV integriert [18]. Da diese Möglichkeit neu ist, konnte sie nicht analysiert werden. Sie bietet jedoch einen Pfad, den zu ergründen sich lohnen könnte. Es ist nicht klar, ob dieses Repository nur den Sourcecode von OpenVino enthält oder eine auf OpenCV spezialisierte Lösung darstellt.

### 6.11.6 Multithreading

Es ist denkbar, die aktuell noch seriell ausgeführten Aufrufe für die Personendetektion zu parallelisieren. Dazu könnte pro zu analysierendes Bild ein Thread erstellt werden. Damit liesse sich die Dauer der Personenerkennung wohl dritteln. Die Lösung besteht bereits auf einem Git-Branch, wurde jedoch aus zeitlichen Überlegungen nicht finalisiert. Denn obwohl die Dauer deutlich reduziert werden könnte, würde nicht einmal eine Drittteilung der Dauer dazu führen, dass eine Echtzeitverarbeitung realistisch würde. Diese Verbesserungsoption muss als Möglichkeit für die Zukunft offen gehalten werden.

## 6.12 Verbesserungen

Es wurden im Laufe des Projekts sehr viele Anpassungen an der Applikation vorgenommen. Es ist zwar nicht möglich, alle Verbesserungen aufzuführen, doch einige wichtige Anpassungen sind es wert, hier erwähnt zu werden.

### **6.12.1 Implementierte Verbesserungen**

Hier sind einige Verbesserungen aufgeführt, welche im Verlaufe des Projektes implementiert wurden.

#### **6.12.1.1 Kameraposition**

Nachdem die erste Version des Trackings umgesetzt wurde, konnte bei der Fehleranalyse festgestellt werden, dass viele Probleme auf eine ungenaue Positionsbestimmung zurückzuführen waren. Eine Kamera war zu tief montiert und konnte dementsprechend zwar Personen erkennen, aber positionierte diese am falschen Ort. Zwei Kameras haben das gesamte Feld überblickt. Das zog nach sich, dass die perspektivische Verzerrung gezwungenermassen grösser wurde. Das wirkte sich auch negativ auf die Positionsbestimmung aus. Durch den Zusatzaufwand einer neuen Aufnahme konnte mit einer verbesserten Positionierung und Ausrichtung der Kameras einiges an Genauigkeit gewonnen werden. Situationen, die bei den ersten Aufnahmen sofort zum Vertauschen von Spielern führten, werden nun problemlos behandelt.

#### **6.12.1.2 Dreieckslösung**

Eine weitere Verbesserung bei der Positionsbestimmung war die Einführung der Dreieckslösung. Initial wurden für die Positions berechnung einfach die im Perspektivenbild am nächsten stehenden Referenzpunkte ausgewählt. Aufgrund der Perspektive sind das aber nicht zwingend die optimalen Punkte, um eine Annäherung mit der baryzentrischen Methode vorzunehmen. Deswegen wurde das Feld in Dreiecke aufgeteilt und wenn ein Spielerpunkt aus der Perspektive ins Modell umgewandelt werden soll, wird zuerst geprüft, ob der Punkt in einem Dreieck liegt. Wenn ja, sind die Referenzpunkte, welche das Dreieck bilden, die optimalen Punkte zur Annäherung der Position. Diese Verbesserung hat viele Probleme gelöst. Mit der alten Methode wurden manchmal unglückliche Referenzpunkt auswählen getroffen, die zu Genauigkeitseinbussen geführt haben.

#### **6.12.1.3 Ausschluss von ungünstigen Konstellationen**

Wird mit der Dreieckslösung kein den Spielerpunkt beinhaltendes Dreieck gefunden, muss auf die ursprüngliche Lösung zurückgegriffen werden. Diese wählt die Referenzpunkte anhand ihrer Nähe zum Spieler in der Perspektive aus. Es gibt viele Konstellationen, welche für eine baryzentrische Berechnung sehr ungünstig sind. Konkret ist es problematisch, wenn die Punkte auf einer Linie oder nahezu auf einer Linie liegen. Somit werden Konstellationen nicht akzeptiert, bei welchen alle Punkte auf derselben Linie liegen. Zusätzlich können auch Konstellation spezifisch ausgeschlossen werden. Diese Verbesserung wurde mit der Einführung der Dreieckslösung etwas abgeschwächt, da nicht mehr viele Positionen auf die ursprüngliche Lösung zurückgreifen müssen. Trotzdem ist es eine Konfigurationsanpassung, welche nützlich sein kann.

#### **6.12.1.4 Suchradius iterativ vergrössern**

Durch den bereits erwähnten Zuletzt-Gesehen-Zähler wurde eine Möglichkeit eingefügt, den Suchradius iterativ zu vergrössern, wenn ein Spieler nicht gefunden wurde. Zudem können gut getrackte Spieler bevorzugt werden.

#### **6.12.1.5 Korrekturmöglichkeit**

Durch die Einführung einer Möglichkeit, Korrekturen vorzunehmen, konnte das Tracking verbessert werden. Werden zwei Spieler vertauscht, funktioniert das Tracking zwar weiterhin, doch die Positionsdaten sind durchwegs falsch. Deswegen wurde die Korrekturfunktion eingeführt.

#### **6.12.1.6 Tracking-Input nicht beschränken**

Als Input für den Tracking-Algorithmus werden aktuell alle von den Kameras erkannten Spielerpunkte mitgeliefert. Ursprünglich wurde der Input zuerst einer weiteren Verarbeitung übergeben. Bei dieser Verarbeitung wurde versucht, alle erkannten Spielerpunkte auf sechs Spieler zu reduzieren. Die Grundidee war, dass, wenn mehrere Spielerpunkte in einem kleinen Gebiet erkannt wurden, dies derselbe Spieler sein muss. Als Beispiel könnte Spieler A von Kamera 1 und Kamera 3 erkannt werden, jede Kamera berechnet die Position und liefert einen Input-Spieler. Da beide Input-Spieler idealerweise sehr nah beieinanderstehen, können sie vereint werden. Somit sollte es als Input für das Tracking für Spieler A nur einen Punkt geben. Das Problem war, dass durch diese Filterung teilweise auch Spielerpunkte verworfen wurden, die eigentlich nützlich gewesen wären. Somit schadete diese Filterung dem Ergebnis und wurde nach Absprache mit Herr Hudritsch verworfen.

## **6.12.2 Verworfene oder zurückgestellte Verbesserungen**

Neben den implementierten Verbesserungen, welche bisher genannt wurden, gab es auch viele Ansätze, welche nach einer Testphase wieder verworfen wurden, weil sie nicht oder nur teilweise ins Konzept passten, die Ergebnisse verschlechterten oder neue Fehlerfälle schufen. Zudem wurden diverse Punkte zurückgestellt, da die Erfolgsaussichten klein oder der Aufwand zu gross für den Umfang dieser Arbeit waren. Die wichtigsten verworfenen Ideen werden hier aufgelistet.

### **6.12.2.1 Dynamischer Radius**

Um zu verhindern, dass zwei Spieler verwechselt werden, wenn sie zu nahe beieinanderstehen, wurde versucht, im Tracking-Algorithmus den Suchradius zu verkleinern. So könnten nur Punkte, die sehr nahe am bisherigen Sichtpunkt des Spielers liegen, als neuer Standort eines Spielers erkannt werden. Das Problem bei diesem Punkt ist es, den Radius passend festzulegen. Ist er zu gross, nehmen die Falschzuweisungen zu, ist er zu klein, werden Spieler verloren, die sich etwas schneller bewegen. Es wurden Versuche mit diversen Radien durchgeführt. Am Ende wurde derjenige, welcher die wenigsten Fehler verursachte, belassen.

Es wurde noch darüber nachgedacht, ob ein dynamischer Radius möglich wäre. Abhängig wäre der Wert von den vergangenen Positionswerten. Haben sich diese in den letzten Frames stark verändert, bewegt sich der Spieler schnell und der Radius sollte gross gewählt werden. Sind nur kleinere Veränderungen feststellbar, sollte auch der Radius klein sein, um Fehlzuweisungen auszuschliessen. Sobald ein Spieler mit einem kleinen Radius nicht mehr erkannt wird, muss schnell reagiert und der Radius drastisch erhöht werden, um ihn wiederzufinden. Diese Überlegungen mussten jedoch aufgrund des Zeitplans zurückgestellt werden.

### **6.12.2.2 Bewegungsvektor**

Um die Verwechslungen zu minimieren, wurde in Betracht gezogen, pro Spieler einen Bewegungsvektor zu führen, welcher seine aktuelle Laufrichtung beinhaltet. Dieser Vektor kann aus den bekannten vergangenen Positionen des Spielers berechnet werden. Beim Tracking, wenn für die bisher bekannten Spieler versucht wird, aus den neuen Input-Daten eine Position auszuwählen, wurde das Zentrum des Suchkreises nicht bei der letzten bekannten Position des Spielers gesetzt, sondern die letzte bekannte Position wurde um den Bewegungsvektor erweitert und der entstehende Punkt als Zentrum genommen. Auch diese Lösung hat sich leider als nicht praxistauglich herausgestellt, da viele neue Fehler entstanden sind. So wurden zum Beispiel Spieler, welche schnell gerannt sind und anschliessend abrupt stoppten, verloren.

Schlussendlich erwies sich die in diesem Dokument beschriebene Lösung als die robusteste.

### **6.12.2.3 Performance**

Die Performance-Tests waren eine weitere zurückgestellte Verbesserung. So wurden diverse Optionen getestet und viel Zeit mit Recherche und Tests verbracht. Die Ergebnisse wurden mit Referenzen belegt.

### **6.12.2.4 Feinjustierung der Parameter**

Die Feinjustierung der diversen Parameter konnte nicht abschliessend durchgeführt werden. Es gibt in dieser Applikation sehr viele Optionen. Die dokumentierte Lösung entspricht der besten bisher gefundenen Konfiguration des Gesamtsystems. Verbesserungen sind aber sicherlich möglich.

### **6.12.2.5 Multithreading**

Das Multithreading wurde initial implementiert und schlussendlich aufgrund mangelnder Zeit und des fehlenden Nutzens verworfen.

### **6.12.2.6 Automatische Fehlerkorrektur**

Die Basis für eine automatische Fehlererkennung und Korrektur wurde in Matlab umgesetzt. Mit diesem Ansatz sind deutliche Verbesserungen denkbar. Zudem sollte der Operator überflüssig werden.

### **6.12.2.7 Farberkennung**

Die Farberkennung hat noch Potential. Für dieses Projekt wurde beschlossen, sich auf eine Unterscheidung von Rot und Gelb zu beschränken.

### **6.12.3 Nicht angegangene Verbesserungen**

Es gibt noch eine dritte Kategorie an Verbesserungen: Diejenigen, welche noch nicht ausprobiert wurden. Es fand, nachdem die erste Version entwickelt war, eine Priorisierung aller Verbesserungsideen statt. Die meisten auf der Liste konnten umgesetzt werden. Einige haben es jedoch aus Zeitgründen nicht in die Implementierungsphase geschafft. Außerdem wurden im weiteren Verlauf des Projektes noch einige mögliche Verbesserungen erkannt.

Die Punkte in der untenstehenden Liste können als eine nicht abschliessende Aufzählung von in der Zukunft auszuprobierenden Lösungsansätzen gesehen werden.

#### **6.12.3.1 Mehr Kameras mit steilem Blickwinkel**

Mit den zweiten selbstgemachten Spielaufnahmen konnten bereits einige Verbesserungen bezüglich Kameraposition erzielt werden. Im letzten Teil des Projektes wurde aber klar, dass noch viel mehr gemacht werden könnte. Dadurch wurde ein Idealbild einer Aufnahme geschaffen, welches im Projekt aus zeitlichen und finanziellen Gründen nicht umgesetzt werden konnte.

Die ideale Aufnahme besteht nach dem aktuellen Wissensstand aus etwa 10 Kameras, wobei jede Kamera nur einen sehr kleinen Teil des Spielfeldes abdeckt. Überschneidungen sind eher kontraproduktiv. Die Kameras sollten von oben auf das Spielfeld hinabschauen und eine kleine Verzerrung aufweisen. In den aktuell verwendeten Aufnahmen wurde ein sehr grosser Sichtwinkel gewählt, was zu Verzerrungen führt. Diese Verzerrungen führen dann wiederum zu Fehlern in der Positionsbestimmung.

#### **6.12.3.2 Kamerasynchronisation**

Die Kamerasynchronisation funktioniert aktuell nicht schlecht. Verbesserungen sind aber sicherlich denkbar und möglich.

#### **6.12.3.3 Automatisches Erkennen der Referenzpunkte**

Es ist denkbar, dass die Konfiguration der Referenzpunkte für jede Kamera überflüssig wird. So ist es denkbar, dass die Referenzpunkte automatisch detektiert werden.

## 6.13 Entscheidungen

Wichtige und grundsätzliche Entscheidungen wurden in der folgenden Tabelle dokumentiert.

Entscheidung	Datum	Ausführliche Begründung
Mehr Frames verwenden	12.11.2018	Statt wie vorgesehen zwei Bilder pro Kamera pro Sekunde werden zehn verwendet, um das Tracking zu verbessern. Das zieht Performance-Auswirkungen nach sich.
Neuerkennung/Löschen	19.11.2018	Als Einschränkung wurde entschieden, bei den neuen Aufnahmen keine Spielerwechsel zuzulassen. Somit sind durchgängig dieselben Spieler sichtbar. Es ist also nicht nötig, Spieler während dem Tracking zu vergessen oder welche neu zu erfassen.
Ausnahmefall bei Farberkennung	26.11.2018	Bei den neusten Aufnahmen hatte eine Kamera zu dunkle Bilder aufgenommen. Da die schnellste Lösung eine Anpassung der Farberkennung war, wurde eine künstliche Aufhellung des Videos gar nicht in Betracht gezogen.
Verzicht auf Parallelisierung des YOLO-Aufrufs	8.12.2018	Es wäre möglich, die zeitintensiven YOLO-Aufrufe zu parallelisieren. Das wurde auch nahezu fertig umgesetzt. Die Performance-Gewinne hielten sich jedoch entgegen der theoretischen Erwartung in Grenzen. Zudem gilt das Argument, dass eine Drittteilung der Ausführungszeit immer noch zu wenig wäre und dementsprechend nicht umgesetzt zu werden braucht.
Auslagern der Demo	18.12.2018	Aufgrund der Performanceprobleme ist eine Live-Demo nicht machbar. Dementsprechend wurde entschieden, die Ergebnisse mit einem Video zu demonstrieren. Die für das Video verwendeten Daten sind jedoch echt.

Tabelle 11 Wichtige Entscheidungen, die während des Projekts getroffen wurden.

## 6.14 Fehlerfälle im Endprodukt

In diesem Unterkapitel soll eine Übersicht gegeben werden, welche Teile des Ablaufes der Applikation fehlerhafte Daten an nachfolgende Schritte liefern. Diese Auswertung beruht auf fünf Frames, welche jeweils acht Sekunden voneinander entfernt sind. Die Fehler werden pro Unterkapitel beschrieben und quantifiziert. Die Daten können nicht absolut korrekt angegeben werden, wurden aber von Auge erfasst und bieten eine Informationsgrundlage.

### 6.14.1 Bilder auslesen

Beim Auslesen der Bilder kann die fehlende Präzision der Videosynchronisierung zum Problem werden. Das bedeutet, dass z.B. ein Kamerabild 0,1s nach den anderen beiden aufgenommen wurde. Die Distanz, die von den Spielern in dieser Zeit zurückgelegt wird, führt zu einer Differenz zwischen den Kameras und dementsprechend zu einer Unschärfe in den Positions berechnungen. Ein Beispiel wird in der folgenden Abbildung dargestellt.



Abbildung 32 Der gleiche Spieler zur vermeintlich exakt gleichen Zeit. Es ist ersichtlich, dass der Spieler im rechten Bild schon lossprintet, während er im linken Bild noch beobachtet.

Die Bestimmung des Fehlers ist hier nicht ganz einfach. Er wurde als die Summe aller abweichenden Distanzen pro Spieler definiert. Ist ein Spieler auf einer Kamera nicht sichtbar, ist der Fehler 0.

Frame	Fehler (Meter)								Fehler/Spieler
	Rot #2	Rot #8	Rot #9	Gelb #2	Gelb #8	Gelb #9	Summe		
1	0	0	0	0,3	0,3	0,5	1,1	0,183	
81	0,4	0	0,3	0,7	0,2	0,5	2,1	0,35	
161	0,2	0,3	0	0	0	0,2	0,7	0,117	
241	0	0	0	0	0	0	0,0	0,0	
321	0	0,5	0,3	0,1	0,2	0,1	1,2	0,2	

Tabelle 12 Fehleranalyse beim Auslesen der Bilder.

Grundsätzlich ist aus Entwicklersicht der Fehler durch die Synchronisation vernachlässigbar. Bei schnellen Sprints könnte dieser Fehler jedoch ins Gewicht fallen. Als Beispiel dient die untenstehende Abbildung. Zwei rote Spieler bewegen sich relativ schnell von der rechten unteren Ecke zur linken oberen Ecke der Abbildung. Kamera 1 erkennt Spieler A als 1101. Die Positionsbestimmung ist korrekt. Kamera 2 erkennt Spieler A als 2106. Auch hier wird die Position korrekt bestimmt. Spieler B wird von Kamera 2 als 2102 und von Kamera 3 als 3101 erkannt. Die Positionsbestimmung ist erneut korrekt. Punkt 2102 und 2106 zeigen die Spieler wenige Hundertstelsekunden später als 3101 und 1101.

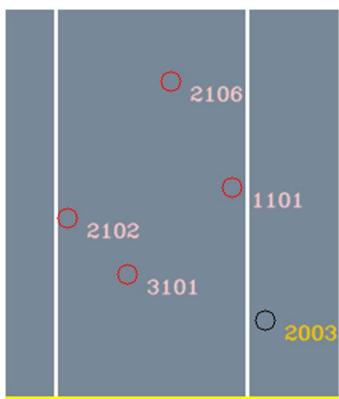


Abbildung 33 Beispiel eines Synchronisationsfehlers.

Somit müsste eine verbesserte Synchronisation in Betracht gezogen werden. Da diese Fehlerquelle jedoch nicht zu vielen Tracking-Fehlern führt und die Zeit knapp war, wurde dieser Punkt zurückgestellt.

#### 6.14.2 Personenerkennung

In diesem Punkt wird einzig und allein die Präzision der Personenerkennung mit YOLO analysiert. Dazu gibt es verschiedene Arten von Fehlern: Ein Spieler kann nicht, doppelt oder sonst fehlerhaft erkannt werden. Dies wird in der nächsten Tabelle analysiert. Bei doppelt erkannten Spielern tritt ein selbst entwickelter Filter in Kraft. Ein Spieler wird in dieser Auswertung nur als Fehler gewertet, wenn er nach der Filterung immer noch doppelt erkannt wird.

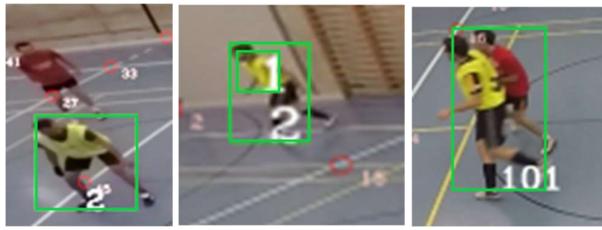


Abbildung 34 Links: Ein unerkannter Spieler. Mitte: Doppelt erkannter Spieler. Rechts: Sonstige Fehlererkennung, welche «Fusion» getauft wurde.

Einzelne fehlende Erkennungen sind nicht sehr schwerwiegend, da die Applikation mit fehlenden Daten umgehen kann. Auch die doppelten Erkennungen sind nicht tragisch, da es nur eine leichte Verstreuung der Resultate bedeutet. Sonstige Fehlererkennungen wie «Fusionen» kommen hin und wieder vor und sind vor allem für die Farbdetektion störend.

Wie fehlende Erkennungen aber doch zu Tracking-Fehlern führen können, ist im Kapitel «Logische Fehler im Tracking» beschrieben.

Frame		Fehler (Anzahl)		
	Nicht erkannte Spieler	Doppelt erkannte Spieler	Sonstige Fehlererkennungen	Summe
1	0	0	0	0
81	0	1	0	1
161	1	0	0	1
241	0	1	0	1
321	3	0	0	3

Tabelle 13 Fehleranalyse der Personenerkennung mit YOLO.

Die von YOLO gelieferte Bounding Box (jeweils auf den Bildern grün eingezeichnet) ist die Grundlage zur Positionsbestimmung. Somit wird eine weitere Analyse gemacht, welche die Summe aller Abweichungen zwischen effektiver Spielerposition und dem Mittelpunkt des unteren Randes der Bounding Box darstellt. Die Distanzen werden geschätzt und in Metern angegeben. Als effektive Spielerposition wird der Punkt der Halle bezeichnet, welcher sich zwischen beiden Füßen des Spielers befindet.



Abbildung 35 Die gelbe Linie stellt die Fehlerdistanz zwischen der effektiven Spielerposition (blauer Punkt) und der vom Algorithmus verwendeten Position (roter Punkt) dar.

Frame	Fehler pro Spieler (Meter)														Summe	Mittelwert Pro erkanntem Spieler
	1	2	3	4	5	6	7	8	9	10	11	12	13	14		
1	0,4	0,6	0,2	0,3	0,1	0,1	0,2	0,0	0,3	0,2	0,2	0,3			2,9	0,242
81	0,3	0,3	0,2	0,1	0,3	0,3	0,1	0,3	0,2	0,0	0,2	0,2			2,5	0,208
161	0,0	0,2	0,0	0,3	0,0	0,1	0,2	0,2	0,5	0,3	0,2	0,0	0,1	0,2	2,3	0,164
241	0,3	0,3	0,0	0,1	0,4	0,4	0,0	0,0	0,1	0,2	0,0	0,0	0,2		2	0,154
321	0,0	0,0	0,3	0,0	0,5	0,4	0,2	0,4	0,3	0,0	0,1	0,0			2,2	0,183

Tabelle 14 Fehleranalyse Bounding Box.

Bei Spielern, welche nahe am Rand der Kamera gefilmt wurden, ist die Abweichung aufgrund des Fischaugeneffektes am Grössten. Der Maximalwert der Abweichungen beträgt aber etwa 60cm, was die aktuelle Methode genügend präzis macht. Trotzdem sind hier Verbesserungen denkbar.

#### 6.14.3 Spielerextraktion

Wenn eine Kamera einen Spieler erkennt, wird seine Trikotfarbe bestimmt. Somit besteht die Möglichkeit zu einer Fehlklassifizierung. Die Fehlerzahl wird als Anzahl Fehlklassifizierungen pro Frame angegeben. Ein Spieler, welcher von mehreren Kameras erkannt wird, kann auch mehrmals falsch klassifiziert werden.

Frame	Fehler (Anzahl)	Anzahl erkannte Spieler	Mittelwert (Fehler/erkannter Spieler)
1	0	12	0
81	0	12	0
161	0	14	0
241	1	13	0,077
321	0	12	0

Tabelle 15 Fehleranalyse bei der Farberkennung.

Die Fehlklassifikation der Farbe ist kein akutes Problem. Wenn gelbe und rote Spieler sich auf engem Raum aufhalten, kann es zu Fehlern kommen. Doch diese sind schwierig zu verhindern.

#### 6.14.4 Positionsbestimmung

Die Positionsbestimmung ist abhängig von der errechneten Bounding Box in der Personenerkennung und der nicht ganz perfekten Synchronisation beim Auslesen der Bilder. Dementsprechend hat der dort bestimmte Fehler auch eine Auswirkung auf die Präzision der Positionsbestimmung. Da diese Fehler aber bereits ausgewiesen wurden, wird hier nur die Summe der Abweichungen zwischen der errechneten und realen Position des Mittelpunkts des unteren Randes der Bounding Box berechnet. So wird gezeigt, wie gross der Fehler bei der Umwandlung zwischen den Koordinatensystemen ist.



Abbildung 36 Obwohl die korrekte Position des Spielers beim roten Punkt läge, wird die berechnete Position (gelber Punkt) als korrekt bewertet, da die Bounding Box relevant ist, nicht die Spielerposition.

Es werden also nur die Fehler bezüglich der gelben Punkte summiert. Die anderen wurden bereits erklärt. Der Wert 0 bedeutet, dass kein sichtbarer Fehler festgestellt werden kann.

Frame	Fehler pro Spieler (Meter)															
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	Summe	Mittelwert Pro erkanntem Spieler
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
81	2,0	0	0	0,3	2,0	0	0	0	0	0	0	0	0	0	4,3	0,358
161	0	0	0	0	0	0	0,5	0	0,7	0	0	0	0	0	1,2	0,086
241	0	0	0	0	0	0	9,0	0	0	0	0	0	0	0	9	0,692
321	0	0	0	0	0,3	0	0	0	0	0	0	0	0	0	0,3	0,025

Tabelle 16 Fehleranalyse bei der Positions berechnung.

Es ist ersichtlich, dass entweder kein oder ein relativ grosser Fehler besteht. Der Grund für diese Probleme ist, dass bei Spielern in Randgebieten der Kamera meist nicht auf mittels Dreieckslösung eruierte Referenzpunkte zurückgegriffen werden kann. Und die Alternativlösung zur Bestimmung der Referenzpunkte ist in Randgebieten ebenfalls nicht gut, da in diesen Situationen oft schlechte Kombinationen vorgeschlagen werden.

Es ist aber erfreulich zu sehen, dass die meisten Positions berechnungen (55 von 62) keinen sichtbaren Fehler aufweisen. Vier Fehler sind vernachlässigbar klein (unter einem Meter) und nur drei sind problematisch. Aber auch die grossen Abweichungen führen nicht unbedingt zu einem Fehler im Tracking.

## **6.15 Logische Fehler im Tracking**

Die Fehler im Tracking werden gesondert in diesem Kapitel beschrieben. Der Grund dafür ist, dass das Tracking-Modul der letzte Schritt in der Verarbeitungskette ist und die dort sichtbaren Fehler aus den vorhergehenden Fehlern entstehen.

Nach 50 Sekunden in der Testsequenz passiert eine Verwechslung, bei der die Spieler der roten Mannschaft komplett vertauscht werden. In der Folge wird Frame für Frame gezeigt, wie das passiert. Das soll die Probleme des Systems beispielhaft zeigen. Auf die Analyse der zwei weiteren Vertauschungen wird verzichtet, da dort dieselben Probleme zugrunde liegen.

Das erste Bild ist jeweils eine Kameraperspektive, welche das aktuelle Geschehen zeigt. Die dort angezeigten Nummern werden als «lokale ID» bezeichnet.

Im zweiten Bild sind die aus dem ersten Bild (sowie den beiden anderen Kamerabildern) extrahierten Spieler gezeigt. Personenerkennung und Positions berechnung sind hier bereits erledigt. Das Bild stellt den Input für das Tracking dar. Die dort verwendeten Nummern werden als «Input-Punkte» bezeichnet.

Das dritte Bild ist das Tracking nach der Verarbeitung. Diese Punkte werden als Spieler oder History-Spieler bezeichnet. Die gelben Linien sind die vergangenen Positionen. Hat im Tracking-Bild ein Spieler einen gelben Kreis im Spielerpunkt, heisst das, dass er nicht wiedergefunden und dementsprechend verloren wurde. In jedem zehnten Frame wird die Groundtruth angezeigt. Sichtbar wird sie durch grüne Punkte, welche mit einer Spielernummer versehen sind. Die Spielernummer kann mit den Nummern der getrackten Spieler verglichen werden. Idealerweise liegt also zum Beispiel der rote Tracking-Punkt 107 sehr nah am grünen Groundtruth-Punkt 107.

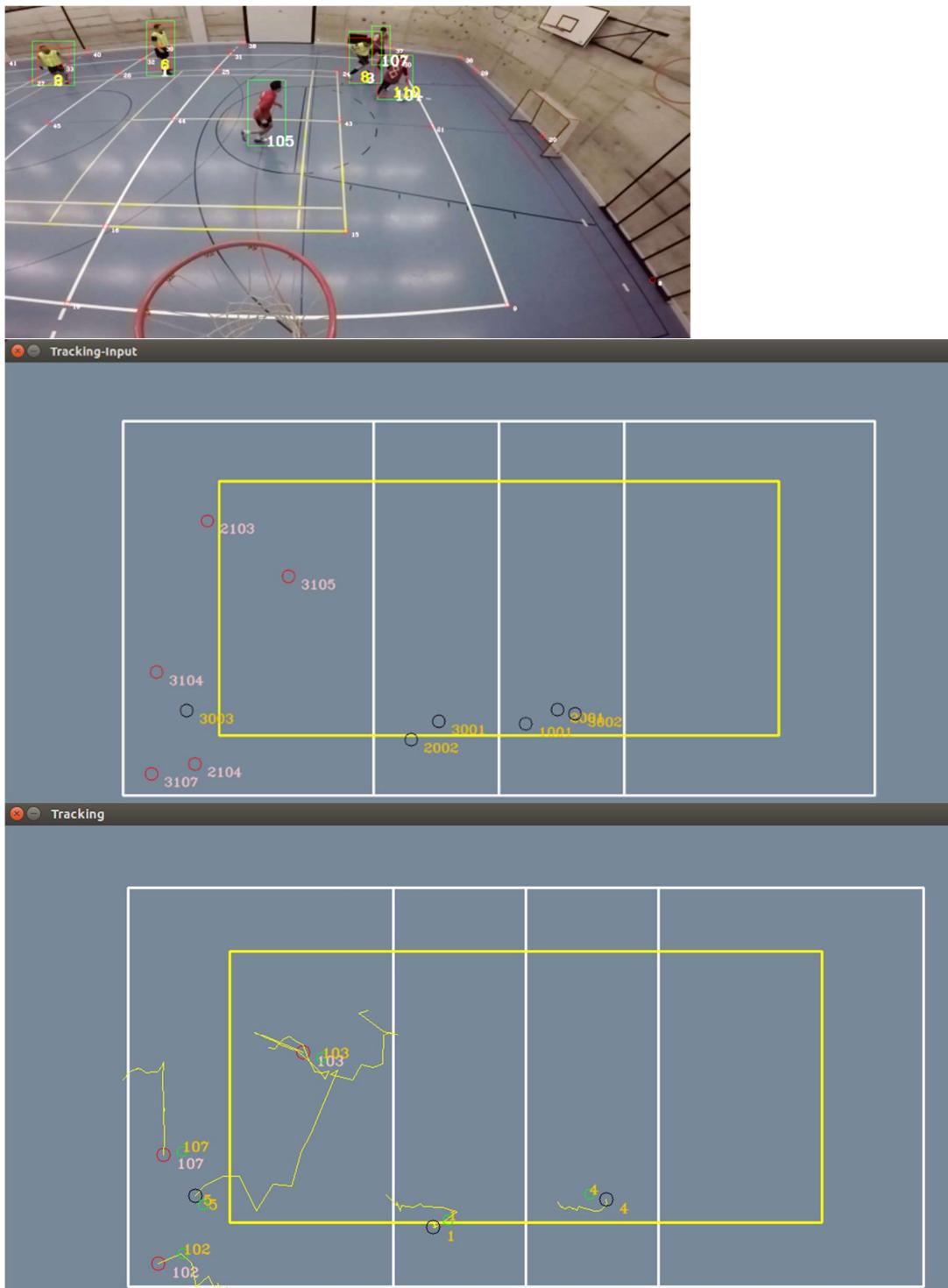


Abbildung 37 Frame 501 der Testsequenz.

Initial ist alles in Ordnung. Die gelbe Mannschaft (schwarze Punkte) greift an und die Positionen der Spieler sind ordnungsgemäss zugeteilt. Input-Punkt 3104 wurde richtig dem Spieler 107 zugeordnet, Spieler 102 konnte sich zwischen den Input-Punkten 2104 und 3107 entscheiden und hat sich für den näheren Punkt 3107 entschieden. Soweit ist alles nach Plan und korrekt.

Die gelben Spieler in der Bildmitte stellen kein Problem dar, weshalb in der Folge nur ein Bildausschnitt gezeigt wird.

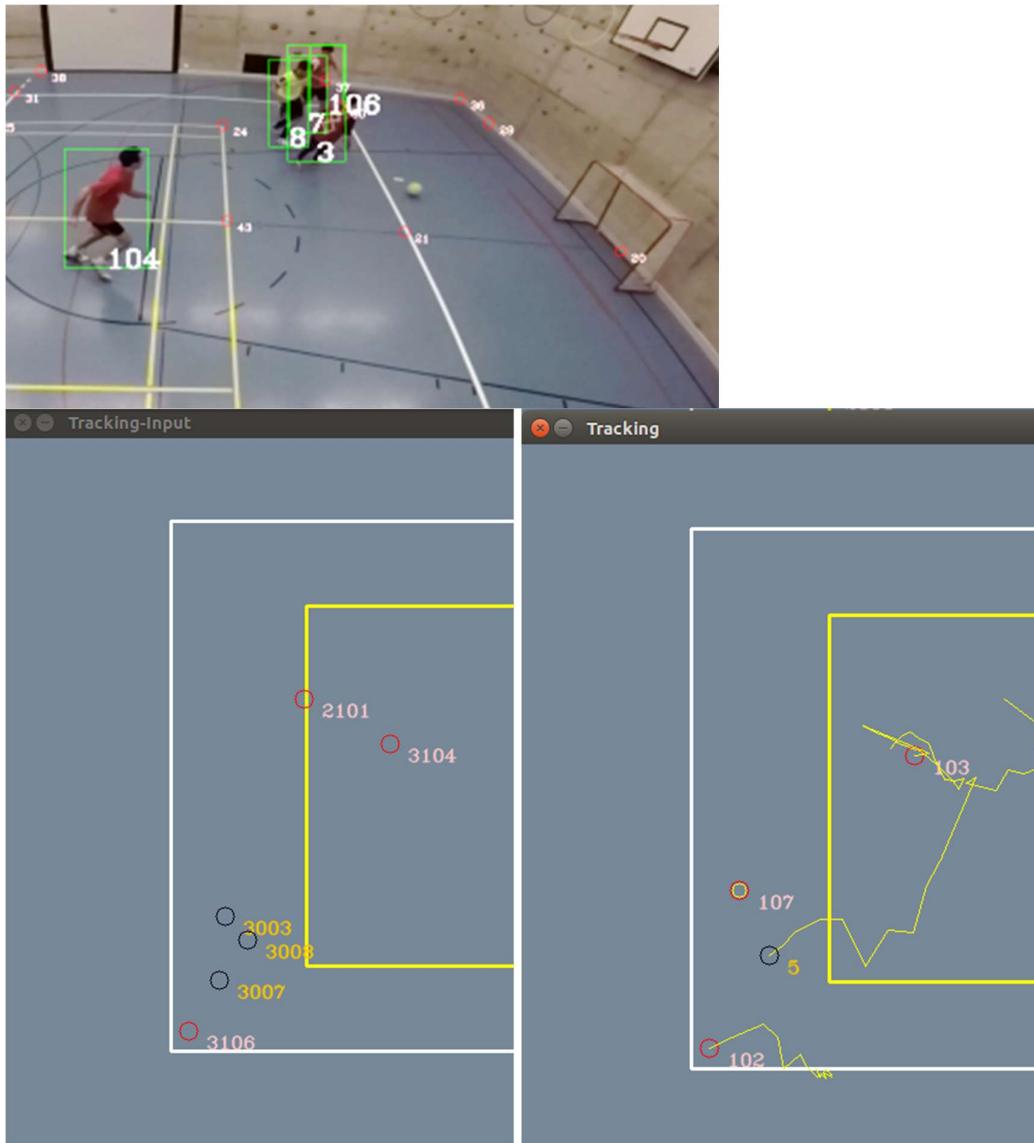


Abbildung 38 Frame 502 der Testsequenz.

Das Übel beginnt nach einer Zehntelsekunde mit einer mangelhaften Personenerkennung von YOLO. Für die drei Spieler 5, 107 und 102 werden insgesamt vier Personen erkannt (Input-IDs 3003, 3007, 3008, 3106). Im Kameraframe werden die Spieler mit der lokalen ID 8 und 106 korrekt erkannt und auch die Farberkennung richtig durchgeführt. Daraus entstehen die Input-Punkte 3008 (gelb, lokale ID 8) und 3106 (rot, lokale ID 106). Zusätzlich wird der mittlere rote Spieler mit der Rückennummer 8 nicht korrekt erkannt und die Bounding Box entspricht einer zu grossen Fläche. Deswegen wird aus der lokalen ID 3 der gelbe Input-Punkt 3003. Dieser Punkt sollte rot sein. Mit der lokalen ID 7 wurde kein einzelner Spieler, sondern eine Mischung aus Spieler 5 und 102 erkannt. Somit wird auch dies fälschlicherweise als der gelbe Input-Punkt 3007 erfasst.

Das alles führt im Endeffekt dazu, dass die Spieler 5 und 102 korrekt zugeordnet werden. Spieler 107 wiederum hat keinen roten Input-Punkt zur Verfügung und kann dementsprechend nicht zugeordnet werden.



Abbildung 39 Frame 503 der Testsequenz.

Auch im nächsten Frame (wiederum eine Zehntelsekunde später) ist die Erkennung unvollständig. Der gelbe Spieler 5 wird wiederum erfolgreich mit der lokalen ID 6 erkannt. Dies ergibt den Input-Punkt 3006. Für den mittleren roten Spieler mit der Rückennummer 8 (entspricht Spieler 107) wird wiederum eine zu grosse Bounding Box angenommen und die Farbe erneut falsch ermittelt. Das resultiert im Input-Punkt 3002. Dieser Input-Punkt wird beim Tracking des Spielers 5 dem Input-Punkt 3006 vorgezogen, weil die Distanz kleiner ist. Mittlerweile werden weder 102 noch 107 getrackt, da keine roten Spieler in dieser Gegend erkannt wurden.

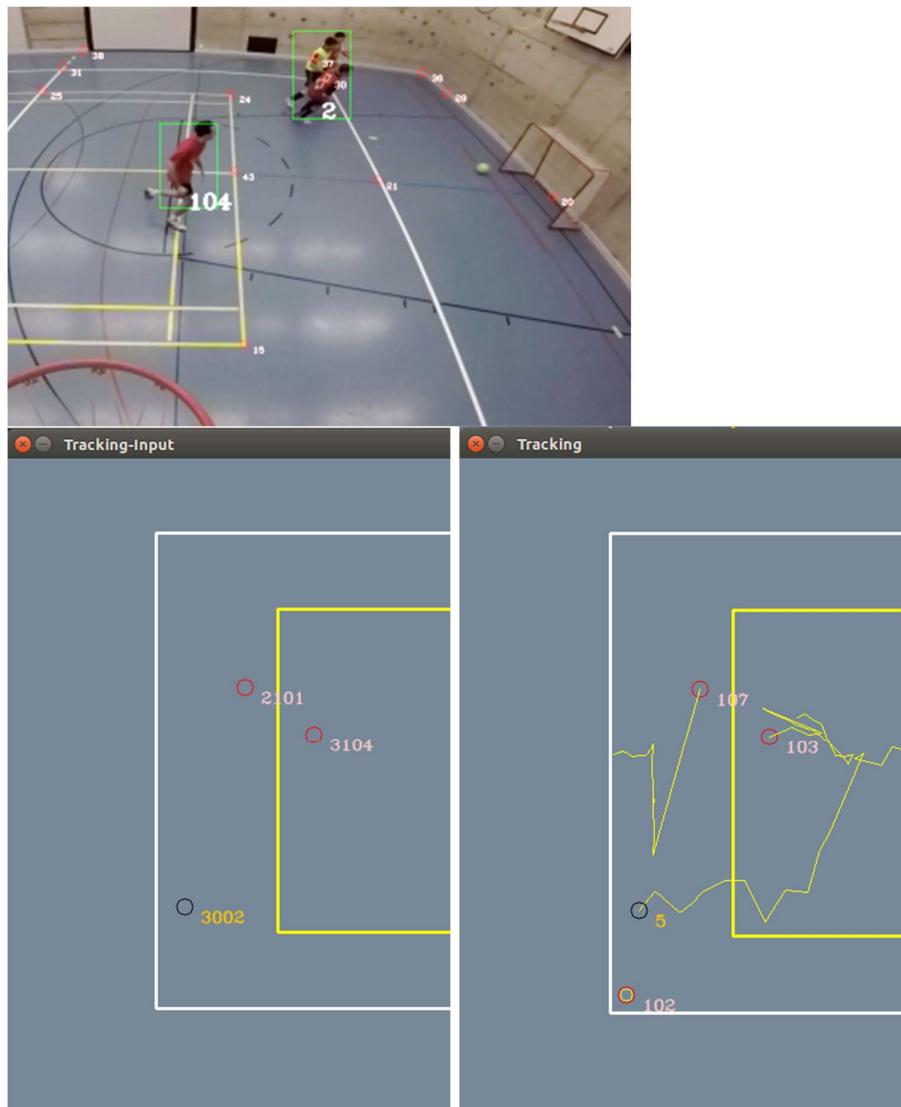


Abbildung 40 Frame 504 der Testsequenz.

Die Situation verschlimmert sich nach einer weiteren Zehntelsekunde erneut: Es wird für die drei Spieler 5, 102 und 107 nur noch ein Input-Spieler erkannt. Die Bounding Box mit der lokalen ID 2 wird zum Input-Punkt 3002. Dieser Punkt wird dem Spieler 5 zugeordnet. Das ist nicht schlecht. Schlecht ist, dass Spieler 107 seit einiger Zeit nicht mehr zugeordnet werden konnte und ihm dementsprechend der Input-Punkt 2101 zugeordnet wird. 2101 entsteht aus einer anderen Kamera, deren Bild hier nicht abgebildet ist. Spieler 102 kann wiederum nicht zugeordnet werden.

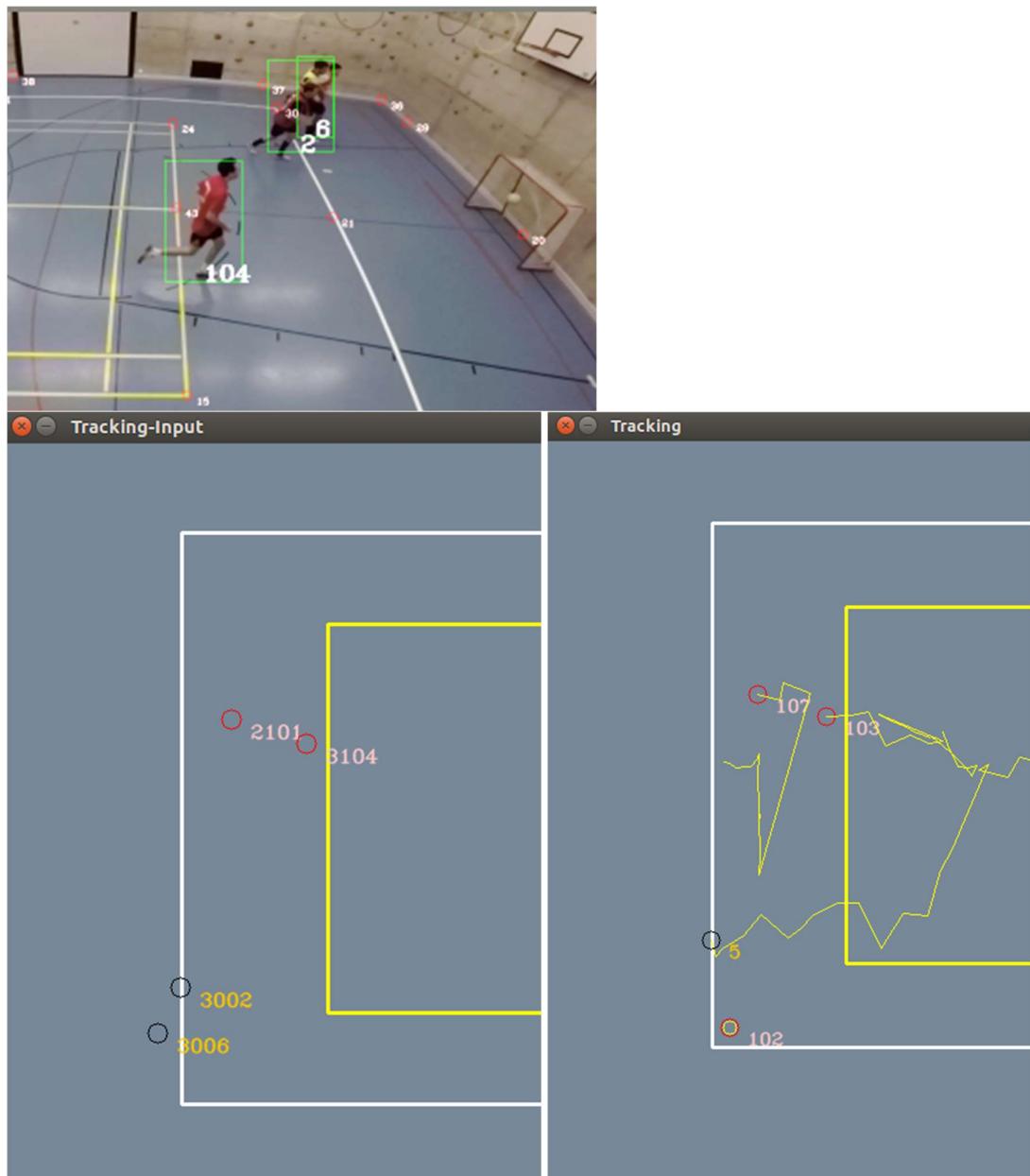


Abbildung 41 Frame 507 der Testsequenz.

Die Situation zieht sich über mehrere Frames mit ähnlichen Erkennungen hin. Hier der Stand nach weiteren drei Zehntelsekunden.



Abbildung 42 Frame 521 der Testsequenz.

21 Frames (2,1 Sekunden), nachdem das Fehlerszenario begonnen hat, verbessert sich die Erkennbarkeit der Spieler etwas. Die zwei oberen roten Spieler werden zwar noch zusammen erkannt, doch immerhin ist kein Teil des gelben Spielers in der Bounding Box vorhanden, womit die Farberkennung nun wieder korrekt funktioniert. Die Zuweisung der Input-Punkte 3102 und 3103 ist der Realität entsprechend. Auch der gelbe Spieler 5 wird immer noch getrackt. Bereits hier ist es, basierend auf den Input-Daten, nicht mehr möglich, definitiv eine Platzierung der Spieler vorzunehmen. Die Situation war zwei Sekunden lang unübersichtlich und es lagen nur mangelhafte Daten vor. In zwei Sekunden ist es einem Spieler möglich, mehrere Meter zurückzulegen und die Position mit einem anderen Spieler zu tauschen.

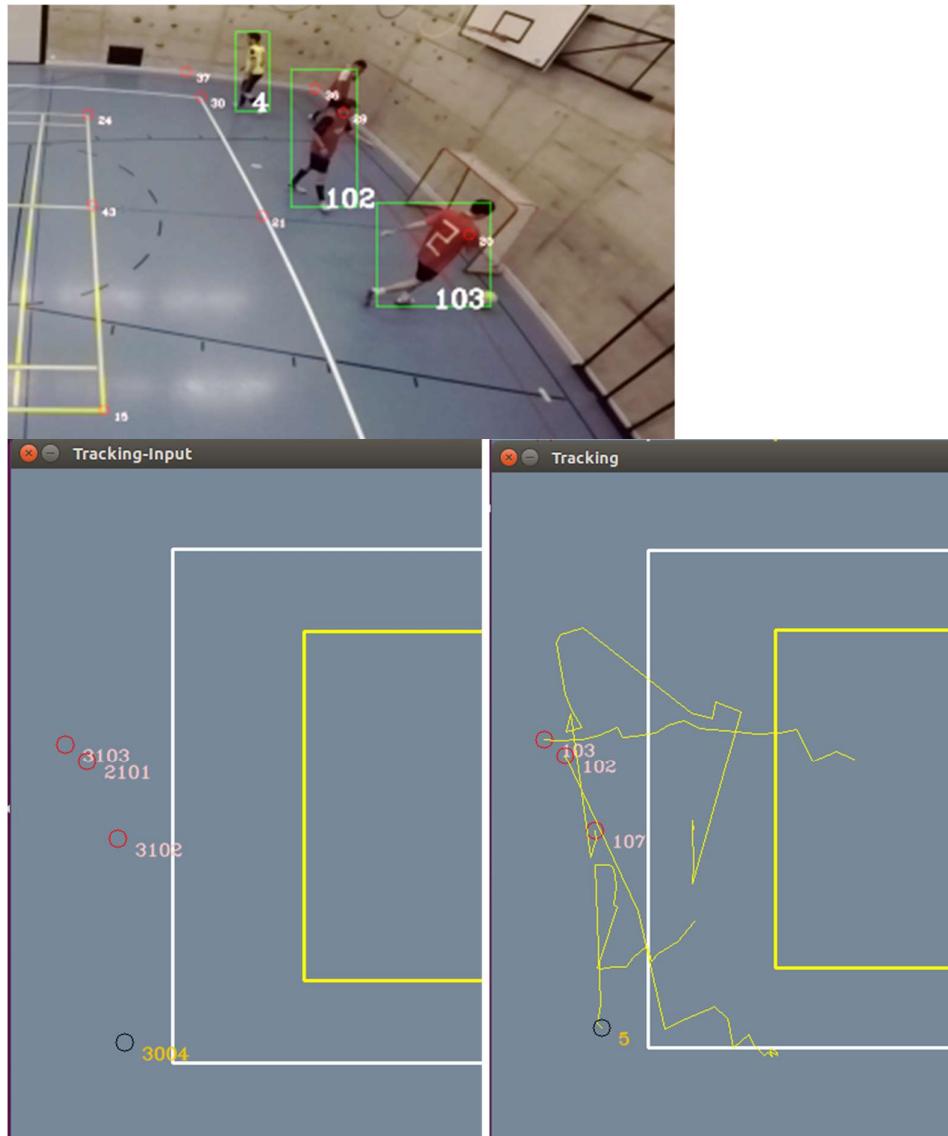


Abbildung 43 Frame 523 der Testsequenz.

Nach weiteren 0,2s zeigt sich das obige Szenario. Da der unterste rote Spieler von mehreren Kameras erkannt wird und dementsprechend die Möglichkeit besteht, dass dort mehrere Spieler stehen, verschiebt sich das Tracking in die Region, in welcher effektiv rote Spieler erkannt werden.

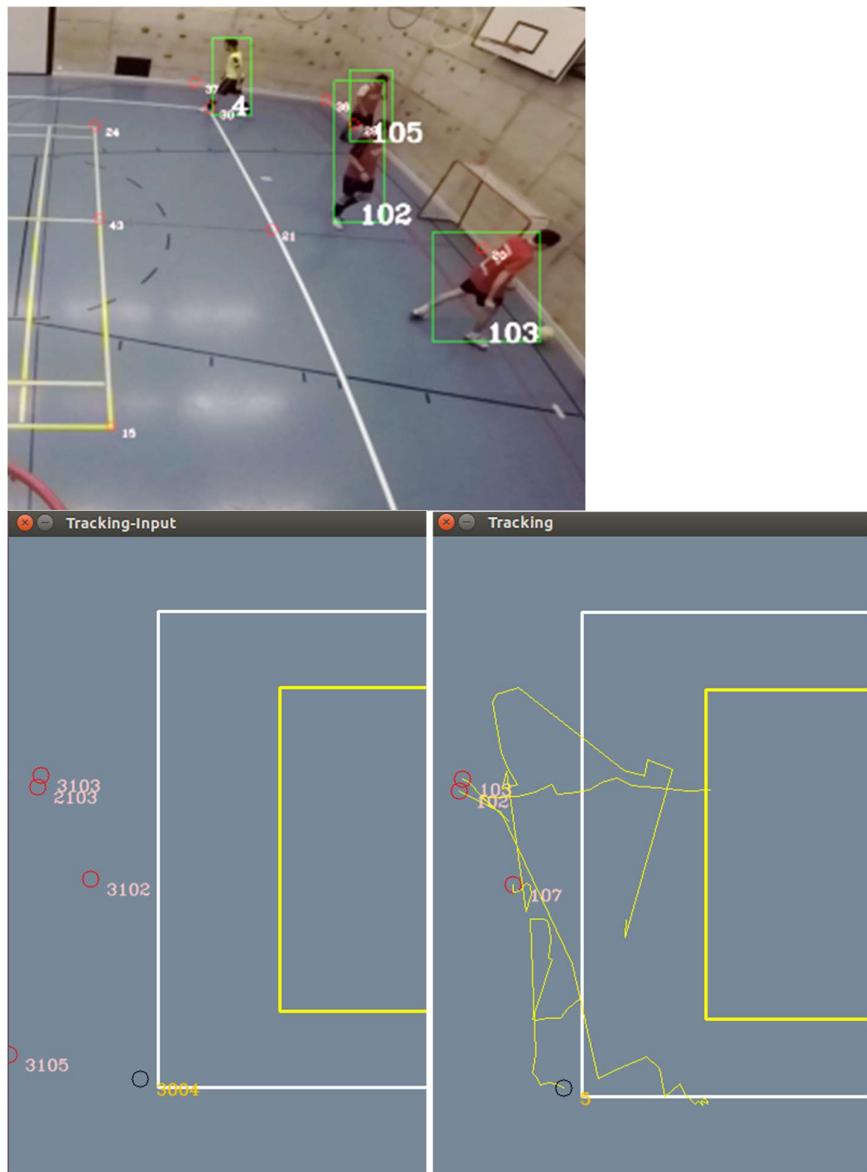


Abbildung 44 Frame 527 der Testsequenz.

Nun (0,4s nach dem letzten Bild) werden wieder alle drei roten Spieler erkannt. Der unterste Spieler wird von zwei Kameras gesehen und bietet dementsprechend zwei rote Input-Punkte an. Das ist auch der Grund, wieso der Input-Punkt 3105 ignoriert wird. Spieler 102 und 103 sind demselben realen Spieler angeheftet.

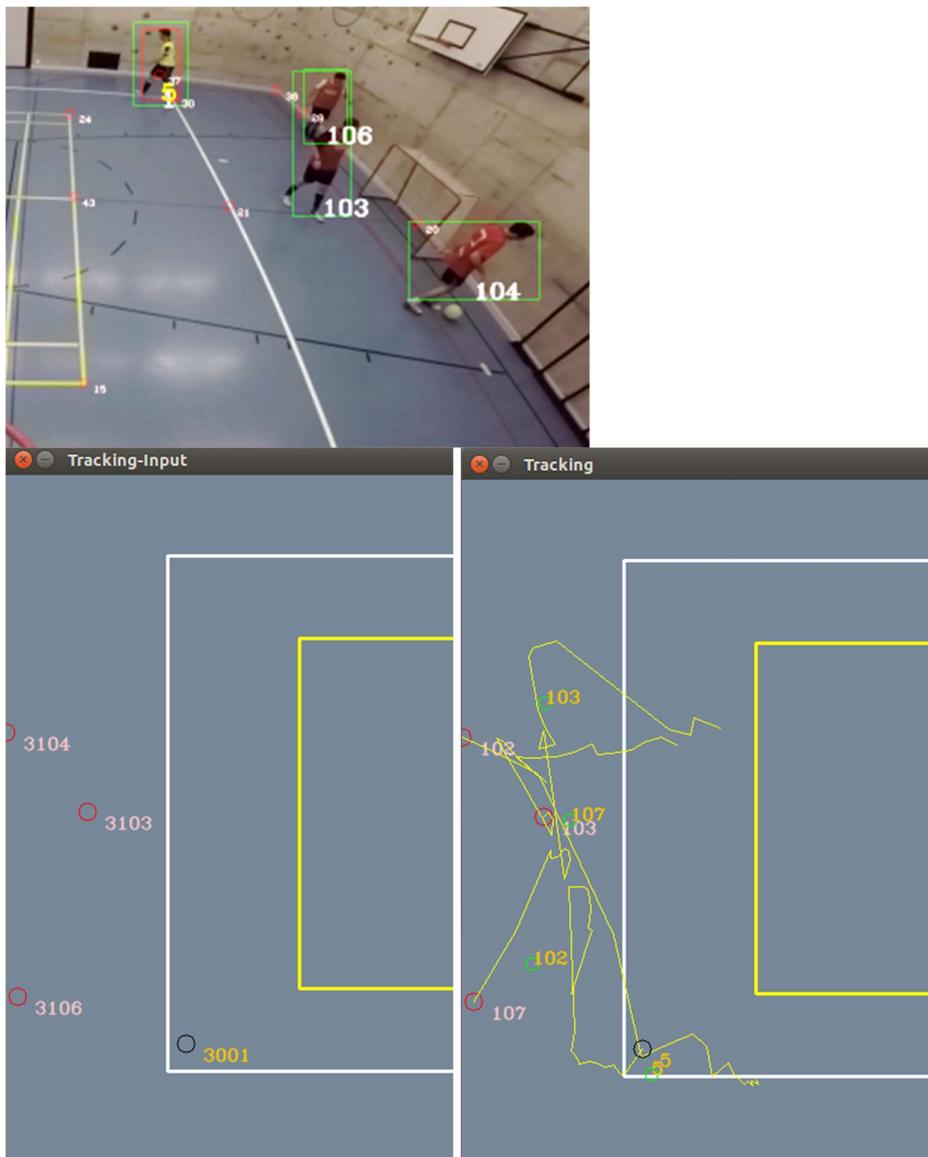


Abbildung 45 Frame 531 der Testsequenz.

In den letzten paar Frames haben sich zwei Spieler an dieselben realen Spieler angeheftet. Nun, nach weiteren 0,3s, teilt sich das Tracking wieder auf. Grund dafür ist, dass nicht mehr genug Input-Punkte verfügbar sind. Unglücklicherweise ist die Zuteilung falsch. Die grünen Punkte im Tracking stellen die Groundtruth dar. Spieler 103 sollte zuoberst sein. In der Mitte sollte 107 stehen und zuunterst wäre 102. Da aber die unübersichtliche Situation etwa drei Sekunden gedauert hat, wurde die Zuordnung verloren. Das hat aber keinen Einfluss auf den gelben Spieler 5. Dessen Zuweisungen waren immer eindeutig, da er der einzige gelbe Spieler in der nahen Region war.

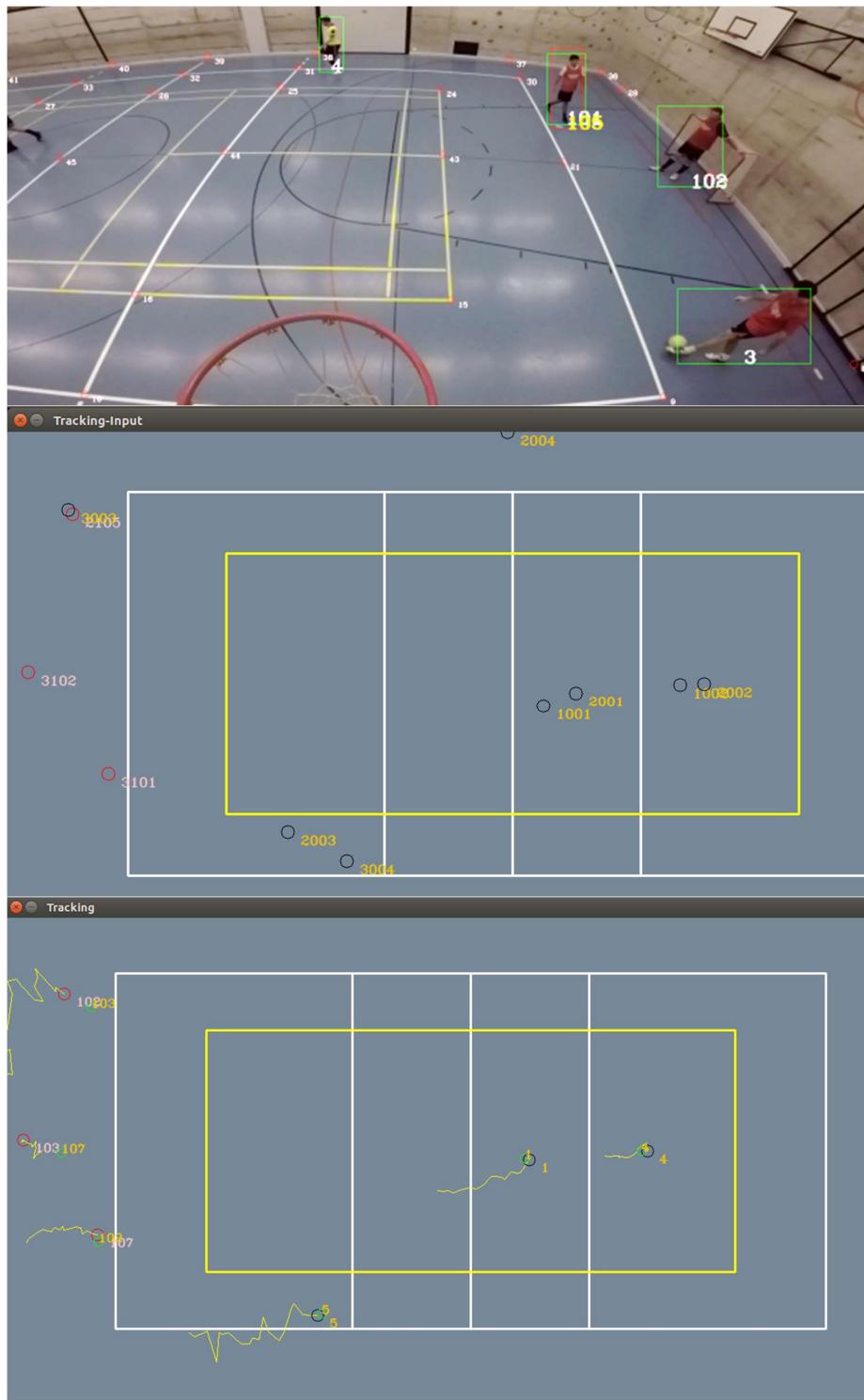


Abbildung 46 Frame 561 der Testsequenz.

Sechs Sekunden nach Beginn der analysierten Situation ist das Spiel wieder übersichtlich. Dementsprechend funktioniert auch das Tracking wieder problemlos. Die Verwechslungen bleiben aber. Hier könnte nun ein Operator eingreifen und die Spieler austauschen.

Wenn die vergangene Sequenz analysiert wird, liegt der Schluss nahe, dass in unübersichtlichen Situationen Verwechslungen von Spielern vorkommen. Die Gründe dazu sind vielseitig und dementsprechend schwierig zu beheben.

Zugrunde liegt das Problem der fehlerhaften Personenerkennung von YOLO. Spieler werden nicht oder falsch erkannt und die Bounding Box umschliesst den erkannten Spieler nicht eng genug. Dies führt dazu, dass die Farbe der Spieler teilweise falsch erkannt wird.

Zudem werden dadurch, dass gewisse Teile des Feldes von mehreren Kameras überblickt werden, aus demselben realen Spieler möglicherweise mehrere Input-Punkte. Die daraus errechnete Position kann aufgrund von Synchronisationsproblemen der Kameras und anderer Kameraperspektive abweichen und eine Unschärfe herstellen.

Auf dieser ungenügenden Datenbasis kann anschliessend kein zuverlässiges Tracking geschehen.

Es sind zwei konkrete Verbesserungen denkbar, welche die beschriebene Situation womöglich besser erkennbar machen liessen:

- **Verbesserung der Farberkennung.** Er wäre denkbar, die Farberkennung noch weiter zu verbessern, damit auch bei mangelhaften Bounding Boxen die dominante Farbe besser erkannt wird. Die Zeit für solche Experimente war in diesem Projekt jedoch nicht mehr vorhanden.
- **Verbesserung der Personenerkennung.** Mit noch mehr Kameras, welche sich auf noch kleinere Bereiche konzentrieren, könnte die Personenerkennung verbessert werden. Im konkreten Beispiel hätte eine Kamera, welche hinter dem Tor befestigt wurde, eine bessere Sicht auf die Szene gehabt und die drei um den Ball kämpfenden Spieler unterscheiden können. Die Kameraposition wurde während des Projekts bereits einmal verbessert, was zu einer deutlichen Steigerung der Genauigkeit des Programms geführt hat. Stünde mehr Zeit zur Verfügung, müssten weitere Aufnahmen gemacht werden. Mehr Kameras und mehr verschiedene Blickwinkel könnten zum Erfolg führen. Das würde natürlich das Performanceproblem verstärken, da die Personenerkennung der Flaschenhals bei der Verarbeitung ist.

Es ist auch zu erwähnen, dass nach einer solchen Situation nicht unbedingt alle Spieler im Tracking zu finden sein müssen. Konkret kann es vorkommen, dass zwei getrackte Spieler in mehreren Punkten (von verschiedenen Kameras) eines Spielers hängen bleiben. Somit hat ein realer Spieler zwei Tracking-Spieler. Die zugrunde liegende Situation ist ähnlich der gezeigten und wird nicht nochmals analysiert. Der Vorgang kann in Abbildung 40 beobachtet werden. Dort sind die Spieler 103 und 107 demselben realen Spieler angehängt.

Zur Behebung solcher Fehler besteht eine Korrektur-Möglichkeit für Operatoren. Diese Art eines menschlichen Eingriffs könnte aber mit der Hilfe von maschinellem Lernen automatisiert werden. Es ist denkbar, dass ein Neuronales Netz die Spieler lernt und anschliessend wiederzuerkennen versucht. Basieren könnte das auf den Bounding Boxen (Spielerbilder), welche aus dem Kamerabild extrahiert werden. Sobald ein Spieler mit hoher Wahrscheinlichkeit wiedererkannt wurde, kann die Information genutzt werden, um sie mit dem Tracking abzulegen. Auch eine verlässliche Nummernerkennung könnte ähnliche Verbesserungen bringen.

In diesem Projekt war aber die Zeit nicht da, um diese Optionen ernsthaft auszuprobieren. Somit müssen diese Ideen zu einem späteren Zeitpunkt geprüft werden.

## 7 Erfüllung der Ziele

### 7.1 Performance

Die Performance ist aktuell der Schwachpunkt der Applikation. Selbst in der besten geprüften Hardware-Konstellation dauert das Verarbeiten von drei Bildern (eines pro Kamera) 0,75s. Wenn also pro Sekunde, wie in den Zielen definiert, pro Kamera zwei Frames verarbeitet werden sollen, beträgt die Gesamtdauer 1,5s. Dazu kommt, dass, um die aktuelle Genauigkeit zu erreichen, zehn Frames pro Kamera pro Sekunde verarbeitet werden. Somit dauert die Verarbeitung von 30 Frames (10fps für drei Kameras) etwa 7,5s.

Für zukünftige Arbeiten müsste die Performance ins Zentrum rücken und weitere Beschleunigungen der Neuronalen Netze in OpenCV sowie alternative Methoden für die Personenerkennung in Betracht gezogen werden. In der aktuellen Version ist das Performanceziel nicht erreicht.

### 7.2 Kosten

Da die Entwicklungskosten nicht gerechnet werden müssen und zwei der drei GoPros kostenfrei ausgeliehen werden konnten, beträgt die Gesamtsumme der Kosten 300 CHF. Ziel waren 900 CHF. Somit wurde das Kostenziel erreicht.

### 7.3 Genauigkeit

Als Basis für die Genauigkeitsberechnungen diente eine Groundtruth, welche für ein Videosegment mit der Dauer von 90 Sekunden selbst erstellt wurde.

Wird eine Verarbeitung ohne Korrekturen betrachtet, beträgt die Übereinstimmung zwischen Groundtruth und Realität mit einer Akzeptanzdistanz von 2m etwas weniger als 69%.

Das Ziel kann aber als erreicht gewertet werden, da mit drei Korrekturen im Verlauf des Segmentes eine Übereinstimmung von über 96% erreicht werden konnte. Die fehlenden 4% finden sich in den Übergangsphasen wieder. Diese entstehen, weil der Operator meist nicht sofort merkt, dass ein Fehler geschah und dementsprechend verzögert korrigiert.

Mit der beispielhaft angegangenen Klassifizierung der Spielerbilder wurde der erste Schritt in Richtung selbstkorrigierendes Tracking gegangen.

### 7.4 Plattformunabhängigkeit

Die Applikation funktioniert erwiesenermassen auf Windows 10, Ubuntu 16.04 und Ubuntu 18.04. Dementsprechend wurden auch die Installationsanleitungen auf diese drei Betriebssysteme ausgelegt. Es ist zu vermuten, dass auch andere Betriebssysteme unterstützt sind.

Ein weiterer positiver Punkt ist, dass die Applikation auf OpenCV 3.4.1 entwickelt wurde und auf OpenCV 3.4.2, 3.4.4 und 4.0.0 mit minimalen Anpassungen funktioniert. Auch wenn das kein Ziel war, erfreut die Zukunftsauglichkeit der Lösung.

### 7.5 Einfache Konfiguration

Das Ausmessen eines neuen Feldes kann, je nach Grösse, bis zu einer Stunde dauern. Das Testfeld konnte jedoch in etwa 15min vermessen werden. Diese Vermessung muss auch nur einmalig ausgeführt werden.

Das Konfigurieren einer Kamera muss, wenn sie nicht fix installiert ist, bei jeder Aufnahme erneut ausgeführt werden. Der Grund ist, dass die Positionen der Referenzpunkte bekannt sein müssen, um die Positionsrechnung vornehmen zu können. Das Konfigurieren dauert pro Kamera ungefähr eine Viertelstunde.

Somit sind die Ziele bezüglich der einfachen Konfiguration erreicht.

## 8 Fazit

Wenn auf ein Projekt zurückgeblickt wird, sieht man immer positive und negative Punkte. So verhält es sich auch bei dieser Arbeit. Um mit einem positiven Ton aufhören zu können, werden zuerst die negativen Punkte angesprochen.

Es gab während des Projekts diverse Momente, wo ein Austausch mit einem im Projekt involvierten Mitstudenten nützlich gewesen wäre. Da diese Thesis eine Einzelarbeit war, konnten Gesprächspartner nicht einfach gefunden werden. Das führte dazu, dass die Arbeit sich manchmal in eine nicht ganz optimale Richtung bewegt hat. Zudem gab es, da ein eigenes Programm ohne externe Anforderungen erstellt wurde, keine genauen Spezifikationen. Jedes Detail der Logik konnte und musste selbst gefunden und implementiert werden. Da im Laufe einer Implementation immer wieder Scheidewege auftreten, wäre es hilfreich gewesen, alle Optionen ausprobieren zu können, um anschliessend eine informierte Entscheidung zu treffen. Allein war das aus zeitlichen Gründen nicht immer möglich. Das führte dazu, dass die finale Lösung, welche hier dokumentiert ist, sicherlich nicht ideal ist. Es gab Designentscheidungen, welche später im Projekt nicht mehr verändert werden konnten und die sich negativ auf die Leistung der Applikation auswirken. Auch konnten aufgrund mangelnder Zeit nicht alle Verbesserungsmöglichkeiten ausprobiert werden.

Etwas ungewohnt war auch, dass sich die Lösung nicht wie eine Businessapplikation entwickeln liess. Businessapplikationen haben einen definierten Zweck. Richtige und falsche Resultate lassen sich klar auseinanderhalten. Bei dieser Thesis wiederum drehte sich der grösste Teil des Projektes um Fehlerminimierung. Der Ansatz war mir bislang noch nicht besonders vertraut.

Ein negativer Punkt war auch die Performance. Da die Applikation in der aktuellen Version für die Verarbeitung von einer Sekunde etwa acht Sekunden braucht, ist die Anwendbarkeit in der Praxis noch nicht gewährleistet. Es war in der begrenzten Zeit des Projektes nicht möglich, die Geschwindigkeit genügend zu verbessern. Somit steht an dieser Front noch etwas Arbeit an. Sobald die Echtzeitfähigkeit gewährleistet ist, kann das Programm aber durchaus als praxistauglich angesehen werden. Zudem wäre die Applikation nach der Implementation der automatischen Fehlerkorrektur mithilfe eines Neuronalen Netzes für eine nachgelagerte Auswertung durchaus geeignet. Dieser Punkt führt zum positiven Teil dieses Fazits.

Es war mit kleinen finanziellen und personellen Mitteln möglich, eine vielseitige Applikation zu entwickeln, welche viele der Anforderungen erfüllt. Nicht erfüllte Ziele sind erklärbar und mit etwas Mehraufwand nachträglich erreichbar. Hilfreich ist dabei besonders der saubere modulare Aufbau der Applikation, welcher das Ersetzen von Teilstücken einfach macht. Die Applikation ist gut dokumentiert und der Code mit Kommentaren versehen und grösstenteils gut lesbar.

Obwohl vor dem Projekt nicht viel mit C++ gearbeitet wurde, konnte der Zugang zur Sprache schnell gefunden werden. Auch wenn sicherlich einige Unschönheiten gefunden werden können, muss der Code nicht versteckt werden.

Durch die Vielseitigkeit der Applikation mussten viele Themengebiete angegangen werden. So wurde zum Beispiel gegen Ende des Projekts auch ein Neuronales Netz für die Klassifizierung der Spielerbilder trainiert. Viele Punkte, welche nicht abschliessend beantwortet werden konnten, wurden immerhin durch auf Tests basierende Erkenntnisse behandelt und mögliche Zukunftsszenarien aufgezeigt.

Die erstellte Lösung kann ein Problem in der echten Welt lösen und wurde von Anfang bis Ende selbst konzipiert und implementiert. Das Projekt war lehrreich, spannend und ich bin stolz auf das Resultat – auch auf dessen Ecken und Kanten!

## 9 Anhang

### 9.1 Benutzerdokumentation

Zur Verwendung der Applikation und der verschiedenen Hilfsmittel wird hiermit eine Benutzerdokumentation zur Verfügung gestellt.

#### 9.1.1 Datenablage

Der Code der Applikationen befindet sich auf Github unter:

<https://github.com/SoullessStone/MultiPersonCameraTrackingPOC>

Dieses Repository kann, sobald das Tool GIT installiert wurde, auf den eigenen Computer geklont werden:

```
git clone https://github.com/SoullessStone/MultiPersonCameraTrackingPOC.git
```

Dieser Befehl kopiert die Applikation, einige Ressourcen und die Hilfsmittel auf den eigenen Computer. Änderungen können gemäss dem Standard-Gitflow vorgenommen werden [19].

Das Erstellen eines neuen Branches, das Hinzufügen der lokalen Änderungen und das Veröffentlichen auf dem zentralen GIT-Server funktioniert wie folgt:

```
cd path-to-project  
git checkout -b neuer-branch-name  
git add .  
git commit -m "Mein Kommentar zu der Änderung"  
git push -u origin neuer-branch-name
```

Anschliessend kann direkt auf der Github-Webseite ein sogenannter Pull Request erstellt werden, welcher anschliessend von der Projektleitung auf den Haupt-Branch «master» gemerget werden kann.

Wenn das Repository geklont wurde, fehlen noch einige Dateien, ohne die die Applikation sofort abstürzt. Die Dateien müssen separat heruntergeladen werden, um das Projekt-Repository nicht unnötig gross werden zu lassen.

Konkret muss die Gewichtsdatei des neuronalen Netzes von der untenstehenden Webseite heruntergeladen und in den Ordner «resources» im Projektverzeichnis kopiert werden:

<https://pjreddie.com/media/files/yolov3.weights>

Die Videodateien des Spiels, welches analysiert werden soll, können auch direkt in den Ordner «resources» kopiert werden.

Damit sind alle Dateien vorhanden, welche für die Ausführung der Applikation benötigt werden.

#### 9.1.2 Verwendete Libraries und Tools

Die Applikation wurde in C++ geschrieben und benötigt zusätzlich ausschliesslich OpenCV.

Zum Bauen der Applikation wird cmake verwendet.

#### 9.1.3 Installation der Libraries

In der Folge wird beschrieben, wie die genannten Libraries und Tools installiert werden können. Da die Art und Weise der Installation sich von Betriebssystem zu Betriebssystem unterscheidet, sind in den folgenden Unterkapiteln die Installationen für Windows und Ubuntu aufgeführt.

##### 9.1.3.1 Windows

Die Installation wurde auf Windows 10 getestet.

Es wurde in groben Zügen der Installationsanleitung von der Webseite LearnOpenCV gefolgt [20]. Da diese Anleitung jedoch Python und C++ unterstützt, werden hier die für dieses Projekt wichtigen Schritte separat dokumentiert.

1. Installation von Visual Studio (getestet wurde das Vorgehen mit der Version 2017)
  - a. Herunterladen der Installationsdatei von  
<https://visualstudio.microsoft.com/de/vs/older-downloads/>
  - b. «Custom Type of installation» wählen
  - c. Unter «Programming Languages» Visual C++ anwählen
  - d. Installation starten
2. Installation von CMake
  - a. Herunterladen der Installationsdatei von <https://cmake.org/download/>
  - b. Während der Installation «Add CMake to the system PATH for the current user» anwählen.
  - c. Installation starten
3. Herunterladen des Codes von OpenCV
  - a. Die Applikation funktioniert auf diesem Stand von OpenCV sicher, weshalb dieser konkrete Link verwendet wird, welcher das Repository zu einem früheren Zeitpunkt zeigt:  
<https://github.com/opencv/opencv/tree/ab2c21e7c14bd99a83628be1ac88c0345634a446>
  - b. Unter «Clone or download» die ZIP-Datei herunterladen
  - c. ZIP-Datei an einem beliebigen Ort entpacken. Das ist der OPENCV\_PATH.
4. Herunterladen des Codes von opencv\_contrib
  - a. Wiederum soll eine bestimmte Version des Codes heruntergeladen werden:  
[https://github.com/opencv/opencv\\_contrib/tree/c818de992e512eb7858c751c7d8200cedebcb40a](https://github.com/opencv/opencv_contrib/tree/c818de992e512eb7858c751c7d8200cedebcb40a)
  - b. Unter «Clone or download» die ZIP-Datei herunterladen
  - c. ZIP-Datei an einem beliebigen Ort entpacken.
5. Mit CMake ein Visual Studio Projekt für OpenCV erstellen
  - a. CMake-GUI öffnen
  - b. Unter «Where is the source code» den Pfad zum opencv-Ordner angeben
  - c. Unter «Where to build the binaries» den Pfad zum opencv-Ordner angeben und ein «/build» hinzufügen
  - d. Auf «Configure» klicken
  - e. Wenn nach dem Generator gefragt wird, kann folgende Auswahl getroffen werden: «Visual Studio 15 2017 Win64»
  - f. Fertigstellen und gegebenenfalls warten, bis die Konfiguration abgeschlossen ist.
  - g. Im roten Bereich einige Anpassungen vornehmen
    - i. «INSTALL\_C\_EXAMPLES» anwählen
    - ii. Den Wert von «OPENCV\_EXTRA\_MODULES\_PATH» auf den Pfad zum entpackten «modules»-Ordner in opencv\_contrib setzen. Beispiel: C:/dev/opencv\_contrib/modules
    - iii. Auf «configure» klicken
    - iv. «BUILD\_opencv\_saliency» abwählen, da dieses Modul unter Windows 10 nicht funktioniert
    - v. Erneut auf «configure» klicken
  - h. Projekt generieren
    - i. Wenn die oberen Schritte erfolgreich verlaufen sind, wird auf «generate» geklickt.
  - i. OpenCV komplizieren
    - i. Eine Kommandozeile öffnen und zum OpenCV-Ordner navigieren
    - ii. In den Ordner «build» wechseln
    - iii. Folgenden Befehl ausführen: cmake.exe --build . --config Release --target INSTALL
  - j. Umgebungsvariablen konfigurieren
    - i. In Windows die Umgebungsvariablen öffnen
    - ii. Unter «System variables» die Variable «path» editieren
    - iii. Den gesamten Pfad zum Ordner «OPENCV\_PATH\build\install\x64\vc15\bin» hinzufügen

- iv. Unter «user variables» eine neue Variable mit dem Namen «OPENCV\_DIR» und dem Wert «OPENCV\_PATH\build\install» erfassen
  - v. Alles speichern und schliessen
- k. Bauen der Lösung
- i. Eine git-bash oder andere Kommandozeile im build-Ordner des Projektes MultiPersonCameraTrackingPOC öffnen.
  - ii. Folgende Befehle ausführen:
    1. cmake -G "Visual Studio 15 2017 Win64" ..
    2. cmake --build . --config Release
  - iii. Anschliessend ist die Lösung bereit für die Ausführung

#### 9.1.3.2 Ubuntu

Die Installation wurde auf Ubuntu 16.04 getestet.

Die folgenden Befehle können einer nach dem anderen in einem Terminal ausgeführt werden. Wenn alles fehlerfrei durchläuft, wurde OpenCV erfolgreich installiert und das Programm sollte gebaut und gestartet werden können.

```
git clone https://github.com/opencv/opencv.git
git clone https://github.com/opencv/opencv_contrib.git
cd opencv_contrib/
git checkout 3.4
cd ..
cd opencv
git checkout 3.4
sudo apt-get install build-essential
sudo apt-get install cmake git libgtk2.0-dev pkg-config libavcodec-dev libavformat-dev libswscale-dev
sudo apt-get install python-dev python-numpy libtbb2 libtbb-dev libjpeg-dev libpng-dev libtiff-dev libjasper-dev libdc1394-22-dev
mkdir build
cd build/
cmake -D CMAKE_BUILD_TYPE=Release -D ENABLE_CXX11=1 -D CMAKE_INSTALL_PREFIX=/usr/local -D
OPENCV_EXTRA_MODULES_PATH=PATH_TO_CONTRIB/opencv_contrib/module ..
make -j4
sudo make install
cd path_to_MultiPersonCameraTrackingPOC/build
cmake ..
make
```

Anstatt des Branches «3.4» könnten auch direkt die Commits verwendet werden. Für opencv wäre das «ab2c21e7c14bd99a83628be1ac88c0345634a446» und für opencv\_contrib «c818de992e512eb7858c751c7d8200cedebcb40a». Damit hat man exakt den Stand von OpenCV, auf dem die Applikation entwickelt wurde.

Anschliessend ist die Installation bereit für die Ausführung [21].

#### 9.1.4 Ausmessen der Halle

Wie in der Dokumentation der Lösung beschrieben, muss ein Spielfeld (im Beispiel jeweils eine Halle) ausgemessen werden. Dies geschieht auf eine beliebige Weise. Idealerweise besteht anschliessend ein massstabsgereuer Plan. Auf diesem Plan markiert der Benutzer anschliessend gut erkennbare Punkte. Damit sind zum Beispiel die Hallenecken, Schnittpunkte von Linien oder sonstige markante Stellen gemeint. Die Position dieser Punkte in Relation zu einer spezifischen «Nullpunkt-Hallenecke» müssen bekannt sein. Die Punkte, welche Referenzpunkte genannt werden, können später für die Konfiguration verwendet werden.

### 9.1.5 Videoaufnahmen

Die Videoaufnahmen werden mit drei Kameras gemacht. Der Grund dafür ist, dass die Applikation aktuell für diese Anzahl ausgelegt ist. Weitere Kameras einzurichten ist jedoch kein Problem.

Es sollte möglichst viel des Bildes der Kamera für das Spielfeld verwendet werden – Wände und die Decke sind wenig interessant und im Grunde genommen verschwendeter Platz. Es muss sichergestellt werden, dass jeder Teil des Feldes von mindestens einer Kamera aufgenommen wird. Es können bessere Resultate erwartet werden, wenn die Kameras von einer erhöhten Position auf das Geschehen herunterschauen. So ist die Positionsbestimmung genauer.

### 9.1.6 Konfiguration im Code

In der Datei `MultiPersonCameraTrackingPOC/src/MultiTracker.cpp` können diverse Anpassungen vorgenommen werden.

Die Kameras können auf folgende Weise konfiguriert werden:

```
Camera cameraHud(1, "../resources/hudritsch_short2.mp4", 0);
Camera cameraMar(2, "../resources/marcos_short2.mp4", 0);
Camera cameraMic(3, "../resources/michel_short2.mp4", 0);
```

Der zweite Parameter referenziert die jeweilige Videodatei, der dritte Parameter dient als Synchronisationshilfe. Startet zum Beispiel die Videodatei «marcos\_short2.mp4» zwei Sekunden vor den anderen beiden Kameras, kann anstatt 0 der Wert 2000 (zwei Sekunden in Millisekunden) mitgegeben werden. Sind die Videodateien bereits synchron, können die Werte so belassen werden.

Ebenfalls im MultiTracker werden zu den Kameras die Referenzpunkte initialisiert. Dazu gibt es pro Kamera eine Initialisierungsfunktion. «cameraHud» wird durch die Methode `initPointPairsHudritsch` konfiguriert. «cameraMar» wird durch die Methode `initPointPairsMarcos` konfiguriert. «cameraMic» wird durch die Methode `initPointPairsMichel` konfiguriert.

Eine jeweilige Konfiguration besteht aus dem Befüllen der zu der Kamera gehörenden Referenzpunkt-Liste (für die CameraHud ist es `referencePointsHud`). Jeder der beim Ausmessen der Halle gefundene Referenzpunkt, welcher auf dem Kamerabild sichtbar ist, kann hier auf dem Kamerabild lokalisiert und referenziert werden. Das Referenzieren funktioniert mit einem PointPair. Der Konstruktor dieser Klasse nimmt als ersten Parameter die Nummer/ID des Referenzpunktes entgegen. Der zweite Parameter entspricht dem x-Wert des Referenzpunktes (Pixels) im Kamera-Perspektiven-Bild. Der dritte Parameter entspricht dem y-Wert des Referenzpunktes (Pixels) im Kamera-Perspektiven-Bild. Der vierte Parameter entspricht dem x-Wert des Referenzpunktes im Modellsystem. Der fünfte Parameter entspricht dem y-Wert des Referenzpunktes im Modellsystem.

```
referencePointsHud.push_back(PointPair(1, 240, 192, 0, 0));
```

Im obigen Beispiel wird der Referenzpunkt 1 für `cameraHud` konfiguriert. Konkret wird ausgesagt, dass sich der Referenzpunkt 1, welcher die Modell-Koordinaten [0, 0] hat, im Kamera-Perspektivenbild im Pixel [240, 192] zu finden ist.

Es müssen nicht alle Punkte konfiguriert werden. Ist ein Punkt nicht sichtbar, wird er einfach weggelassen. Die Kamera-Pixelkoordinaten können mit dem Hilfstoß «position» einfach und schnell ausgelesen werden. Wie das funktioniert, wird in einem nachfolgenden Unterkapitel erklärt.

Es ist wichtig zu verstehen, dass diese Konfiguration jedes Mal neu gemacht werden muss, sobald sich die Sicht oder Position der Kamera verändert. Denn wenn das geschieht, sind die Referenzpunkte nicht mehr am selben Ort und die Positionsbestimmung funktioniert nicht mehr.

Eine weitere Konfiguration kann in der Datei `MultiPersonCameraTrackingPOC/src/PerspectiveToModelMapper.cpp` vorgenommen werden. In dieser Klasse wird die Logik für die Umwandlung von Perspektiven- zu Modell-Koordinaten verwaltet. Angepasst werden muss die Methode `initTriangles`. Basierend auf den erfassten Referenzpunkten beim Ausmessen des Feldes muss dieses Feld hier in Dreiecke aufgeteilt werden. Dies passiert, indem neue Einträge in die Liste «triangles» eingefügt werden. Das eingefügte Objekt hat einen Konstruktor, welcher drei Referenzpunkt-IDs entgegennimmt. Die drei übergebenen Referenzpunkte bilden somit

ein Dreieck. Auf dem massstabsgetreuen Plan können die Dreiecke eingezeichnet und dann im Code erfasst werden. Das muss pro Feld nur einmal geschehen. Untenstehend ein Beispiel, welches definiert, dass die Referenzpunkte 5, 6 und 13 ein Dreieck bilden.

```
triangles.push_back(PointConstellation(5, 6, 13));
```

Idealerweise ist jeder Punkt auf dem Feld in mindestens einem Dreieck vorhanden.

Ebenfalls in der Datei MultiPersonCameraTrackingPOC/src/PerspectiveToModelMapper.cpp können Konstellationen von Referenzpunkten definiert werden, welche nie zur Positionsbestimmung gebraucht werden sollen. Das geschieht in der Methode initBannedConstellations. Grund für einen Ausschluss könnte zum Beispiel sein, dass die Punkte in einer Linie liegen. Diese Konfiguration kann bei schlechten Positionsberechnungen zu Hilfe gezogen werden, indem herausgefunden wird, welche Referenzpunkte verwendet wurden und diese explizit ausgeschlossen werden. Ein Beispiel, welches verhindert, dass die Referenzpunkte 1, 9 und 15 für die Positions berechnung verwendet werden:

```
bannedConstellations.push_back(PointConstellation(1, 9, 15));
```

Für ein neues Feld müssen die bestehenden Einträge gelöscht werden, da die Verteilung der Referenzpunkte unterschiedlich sein wird und dementsprechend nicht dieselben Konstellationen ausgeschlossen werden können.

Die letzte Möglichkeit für eine Konfiguration ist in der Datei MultiPersonCameraTrackingPOC/src/TrackingModule.cpp in der Klasse initBasetruth zu finden. Diese Konfiguration ist optional. Sie bietet die Möglichkeit, zu Testzwecken eine mit dem Hilfstoß «groundtruth» erstellte Groundtruth direkt in den Trackingresultaten anzuzeigen. So könnte während der Entwicklung relativ einfach die Differenz zwischen Resultat und Wahrheit erkannt werden. Die Konfiguration befüllt eine Map mit dem Namen «basetruth». Die Map enthält als Schlüssel die Frame-Nummer und als Inhalt eine Liste von anzuseigenden Punkten. Im folgenden Beispiel werden für das erste Frame sechs Spieler als Groundtruth erfasst. Als Beispiel: Ein Spieler, welcher mit der Nummer 1 angezeigt wird, befindet sich auf dem ersten Frame auf dem Modell-Koordinaten-Punkt [604, 282].

```
std::vector<PointPair> vector1;
vector1.push_back(PointPair(1,604,282,-1,-1));
vector1.push_back(PointPair(107,132,295,-1,-1));
vector1.push_back(PointPair(103,269,381,-1,-1));
vector1.push_back(PointPair(102,289,293,-1,-1));
vector1.push_back(PointPair(4,594,528,-1,-1));
vector1.push_back(PointPair(5,495,57,-1,-1));
basetruth.insert(std::make_pair(1,vector1));
```

### 9.1.7 Ausführen

Um das Programm auszuführen befindet man sich im «build»-Ordner des MultiPersonCameraTrackingPOC-Projekts. Anschliessend kann unter Windows folgender Befehl abgesetzt werden:

```
./Release/MultiTracker.exe -c=../resources/yolov3.cfg -m=../resources/yolov3.weights -classes=../resources/coco.names --scale=0.00392 --width=416 --height=416
```

Unter Linux lautet der Befehl wie folgt:

```
./MultiTracker -c=../resources/yolov3.cfg -m=../resources/yolov3.weights -classes=../resources/coco.names --scale=0.00392 --width=416 --height=416
```

### 9.1.8 Verstehen der angezeigten Bilder

Wenn die Ausführung erfolgreich ist, werden diverse Bilder angezeigt, welche auch bei der Fehlersuche hilfreich sein könnten.

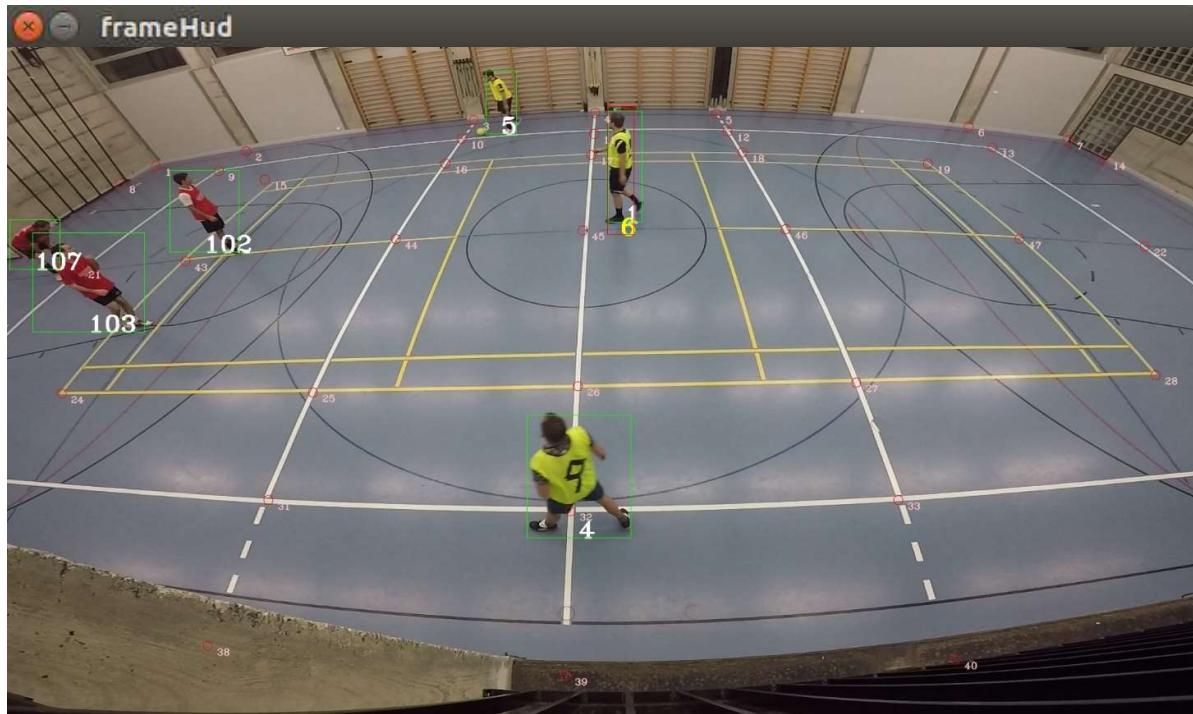


Abbildung 47 Zeigt die Sicht der cameraHud zum aktuellen Zeitpunkt sowie die erkannten Spieler inklusive der für diese Kamera lokalen Spieler-IDs.



Abbildung 48 Zeigt die Sicht der cameraMar zum aktuellen Zeitpunkt sowie die erkannten Spieler inklusive der für diese Kamera lokalen Spieler-IDs. Hier ebenfalls ein Beispiel eines als Duplikat erkannten Spielers (ID 105).

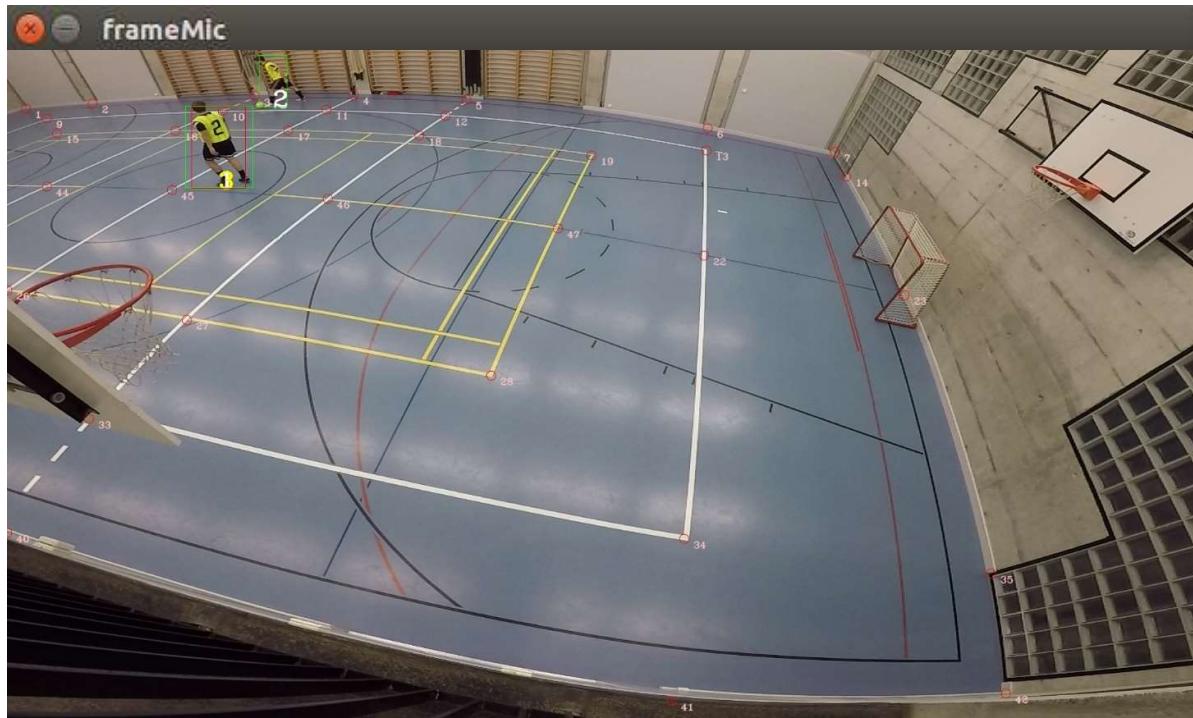


Abbildung 49 Zeigt die Sicht der cameraMic zum aktuellen Zeitpunkt sowie die erkannten Spieler inklusive der für diese Kamera lokalen Spieler-IDs. Hier ebenfalls ein Beispiel eines als Duplikat erkannten Spielers (Spieler mit der Trikotnummer 2).

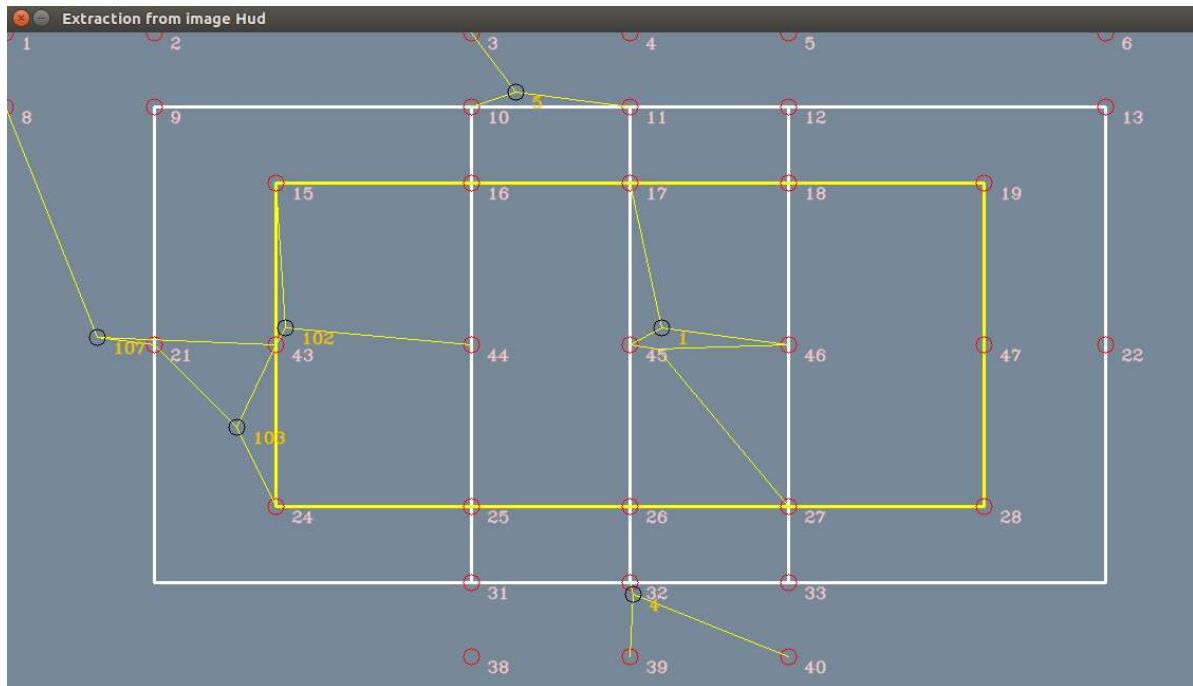


Abbildung 50 Die Kamerasicht von cameraHud übertragen auf das Modell des Feldes. Die Punkte stellen die Spieler dar, die lokale ID des Spielers wird neben dem Punkt angezeigt. Zudem zeigen die gelben Linien für jeden Spieler, welche Referenzpunkte verwendet wurden.

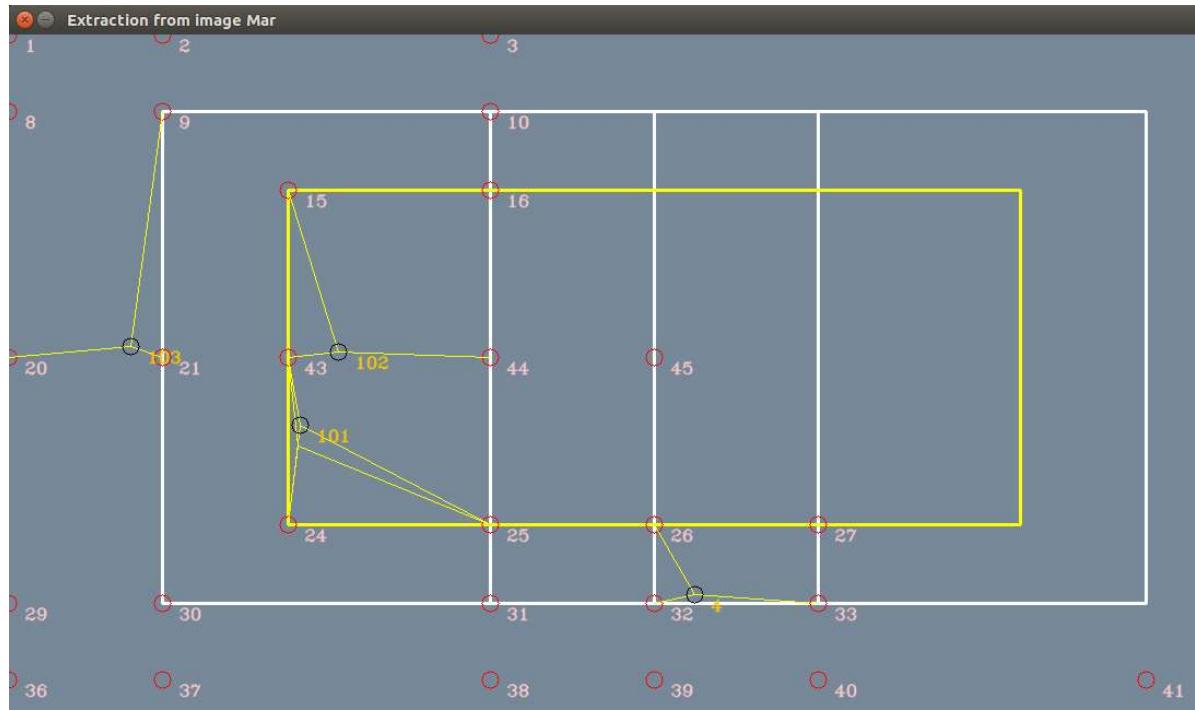


Abbildung 51 Die Kamerasicht von cameraMar übertragen auf das Modell des Feldes. Die Punkte stellen die Spieler dar, die lokale ID des Spielers wird neben dem Punkt angezeigt. Zudem zeigen die gelben Linien für jeden Spieler, welche Referenzpunkte verwendet wurden.

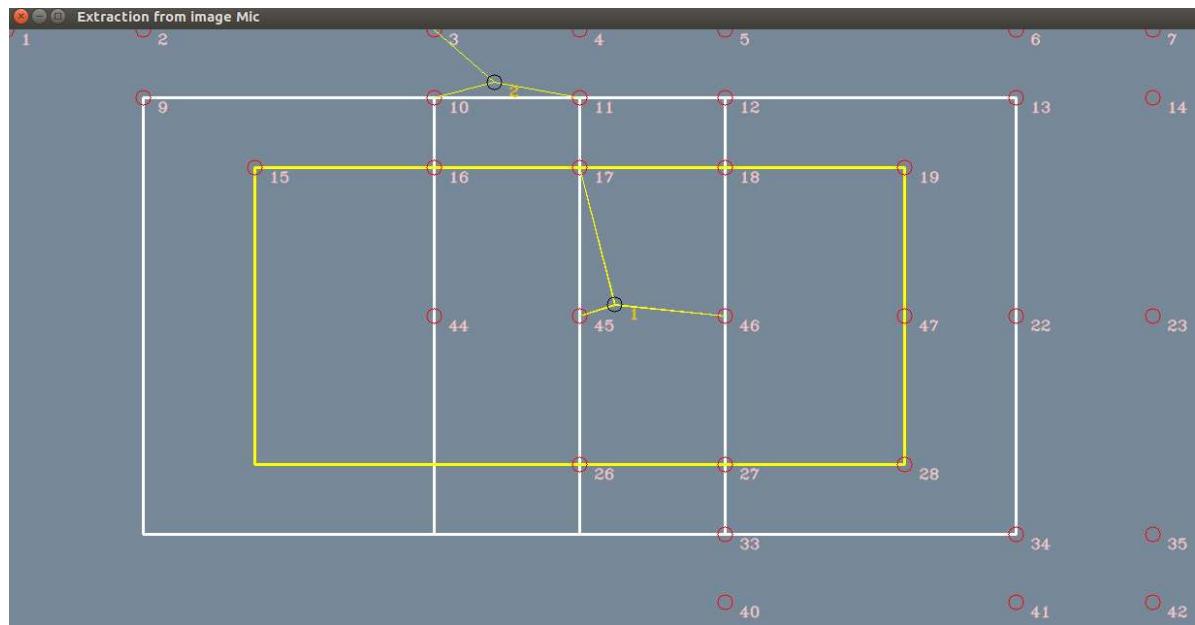


Abbildung 52 Die Kamerasicht von cameraMic übertragen auf das Modell des Feldes. Die Punkte stellen die Spieler dar, die lokale ID des Spielers wird neben dem Punkt angezeigt. Zudem zeigen die gelben Linien für jeden Spieler, welche Referenzpunkte verwendet wurden.

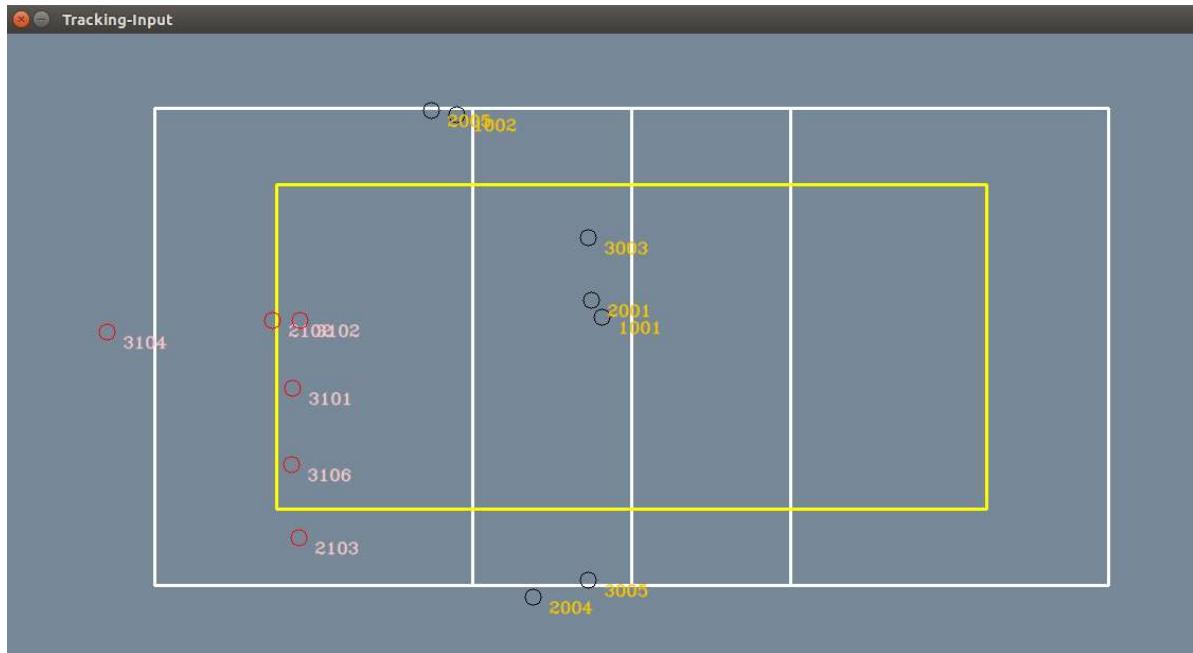


Abbildung 53 Der Input für den Tracking-Algorithmus. Dieser besteht aus allen von den drei Kameras erkannten Punkten in Modell-Koordinaten. Nummern zwischen 1000 und 2000 gehören zu cameraMic, Punkte zwischen 2000 und 3000 kommen von cameraHud, die andern von cameraMar.

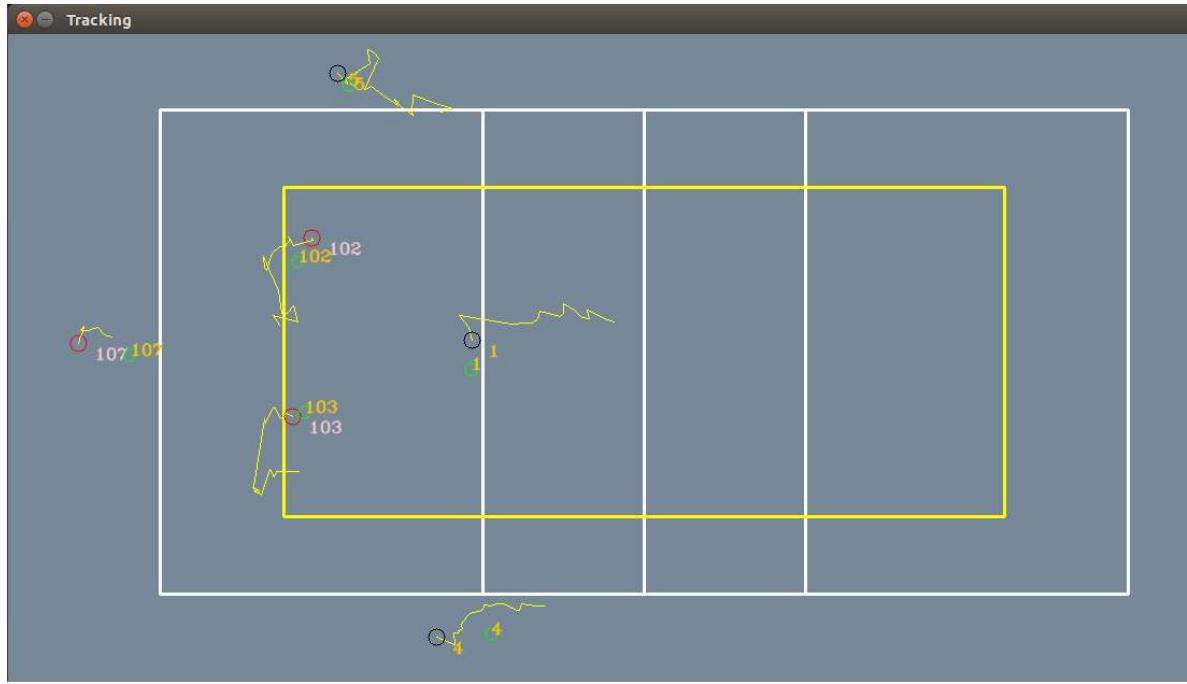


Abbildung 54 Das Trackingresultat, welches die Spieler (rote und schwarze Punkte), deren vergangenen Positionen (gelbe Linien) sowie - falls konfiguriert - die Groundtruth (grüne Punkte) anzeigt.

### 9.1.9 Korrekturfunktion

In der letzten gezeigten Grafik (Tracking) besteht die Möglichkeit bei Bedarf Korrekturen vorzunehmen. Dafür kann bis zu drei Mal in das Bild geklickt werden. Jeder Klick definiert eine neue Spielerposition. Anschliessend wird der Benutzer für jeden Klick gefragt, welcher Spieler sich dort befindet. Ab dem nächsten Frame sind die Korrekturen aktiv.

```
Press Escape to abort the correction.  
Press 1 to assign the clicked point to player #1  
Press 2 to assign the clicked point to player #102  
Press 3 to assign the clicked point to player #103  
Press 4 to assign the clicked point to player #4  
Press 5 to assign the clicked point to player #5  
Press 6 to assign the clicked point to player #107  
Applied correction: 102, 31
```

Abbildung 55 Für jeden Klick in das Tracking-Fenster erscheint eine solche Abfrage im Konsolenfenster.

### 9.1.10 Tools verwenden

Es existieren diverse Hilfsmittel, welche für spezielle Anwendungen im Umgang mit dem Tracking-Programm gedacht sind. In den untenstehenden Unterkapiteln wird die Verwendung auf Ubuntu gezeigt. Auf Windows würden die Tools ebenfalls funktionieren, müssten jedoch analog der Hauptapplikation gebaut und kompiliert werden. Um Redundanzen zu vermeiden, wird das hier nicht mehr explizit dokumentiert.

#### 9.1.10.1 Groundtruth

Dieses Tool befindet sich unter MultiPersonCameraTrackingPOC/resources/tools/groundtruth. Diese Applikation ist dazu da, für Videosequenzen eine Groundtruth zu erstellen. Die Videodateien werden wie folgt (analog Tracking-Applikation) referenziert:

```
Camera cameraHud(1, "../hudritsch_short2.mp4", 0);  
Camera cameraMar(2, "../marcos_short2.mp4", 0);  
Camera cameraMic(3, "../michel_short2.mp4", 0);
```

Die Verwendung ist relativ einfach:

- Öffnen eines Terminals im entsprechenden Ordner
- Ausführen von «cmake ..»
- Ausführen von «make»
- Ausführen von «./Groundtruth Record»

Nun werden die drei Frames sowie ein Modell-Bild angezeigt. Anhand der Kamera-Bilder kann nun für einen Spieler die Position auf dem Feld angeklickt werden. Das führt dazu, dass die Koordinaten in der Konsole ausgegeben werden und die neuen drei Kamerabilder geladen werden. Dabei wird zeitlich eine Sekunde übersprungen. Wie viel Zeit übersprungen werden soll, kann im struct Camera in der Methode getNextFrame angepasst werden, indem auf Zeile 66 eine andere Zahl zu wantedMs addiert wird.

```
wantedMs += 1000;
```

Es ist zu beachten, dass die Ausgabewerte auf der Console noch verdoppelt werden müssen. Grund dafür ist, dass das Tool die Pixelkoordinaten ausgibt und das Feld um den Faktor 0,5 verkleinert dargestellt wird. Somit sind auch die Pixelkoordinaten halb so gross wie benötigt. Das Tool entsprechend anzupassen wäre kein Problem, wurde jedoch aus Kontinuitätsgründen nicht gemacht.

#### 9.1.10.2 PlayerImageOrganizer

Dieses Tool ist unter MultiPersonCameraTrackingPOC/resources/tools/playerimageorganizer zu finden. Es dient dem Klassifizieren von Bildern. Die im Unterordner «players» gespeicherten Bilder werden eines nach dem anderen ausgelesen und können mit einem Tastendruck in die jeweiligen Unterordner verschoben werden. Auf eine genaue Beschreibung dieses Tools wird verzichtet. Grund ist, dass diese Applikation sehr auf die konkreten Bilder ausgerichtet und dementsprechend schwierig wiederzuverwenden ist.

### 9.1.10.3 Position

Dieses Tool ist unter MultiPersonCameraTrackingPOC/resources/tools/position zu finden. Es dient hauptsächlich der schnellen Erfassung der Referenzpunkte für eine Kamera. So muss zuerst ein Frame der zu konfigurierenden Kamera im Ordner gespeichert werden. Dieses wird dann angezeigt.

Die Verwendung ist grösstenteils gleich wie bei der Groundtruth:

- Öffnen eines Terminals im entsprechenden Ordner
- Ausführen von «cmake ..»
- Ausführen von «make»
- Ausführen von «./Position Record bild.png»

Mit diesem Programm wird nun das angegebene Bild angezeigt und es kann ins Bild geklickt werden. Die Koordinaten der angeklickten Referenzpunkte werden dann ausgegeben und können für die Konfiguration der Kamera, von der das Bild stammt, verwendet werden.



Abbildung 56 Ein Beispiel der Erfassung einiger Referenzpunkte.

Die Ausgabe für die obenstehende Abbildung sieht aus wie folgt:

```
Position Record michel.png
1 - Clicked coordinates: 270, 346
2 - Clicked coordinates: 468, 322
3 - Clicked coordinates: 888, 292
4 - Clicked coordinates: 1060, 282
5 - Clicked coordinates: 1226, 280
6 - Clicked coordinates: 1518, 290
7 - Clicked coordinates: 1632, 284
8 - Clicked coordinates: 232, 384
9 - Clicked coordinates: 448, 360
10 - Clicked coordinates: 894, 324
11 - Clicked coordinates: 1086, 312
12 - Clicked coordinates: 1262, 308
13 - Clicked coordinates: 1556, 304
14 - Clicked coordinates: 1670, 308
```

```

15 - Clicked coordinates: 598, 380
16 - Clicked coordinates: 976, 352
17 - Clicked coordinates: 1124, 346
18 - Clicked coordinates: 1264, 334
19 - Clicked coordinates: 1528, 340
20 - Clicked coordinates: 1628, 334
21 - Clicked coordinates: 1758, 330
22 - Clicked coordinates: 554, 516
23 - Clicked coordinates: 276, 548
24 - Clicked coordinates: 954, 480
25 - Clicked coordinates: 1232, 448
26 - Clicked coordinates: 1484, 436
27 - Clicked coordinates: 1696, 418
28 - Clicked coordinates: 1916, 548
29 - Clicked coordinates: 1868, 706
30 - Clicked coordinates: 1710, 598
31 - Clicked coordinates: 1420, 650
32 - Clicked coordinates: 1560, 796
33 - Clicked coordinates: 1100, 900
34 - Clicked coordinates: 1026, 726
35 - Clicked coordinates: 444, 780
36 - Clicked coordinates: 98, 786
37 - Clicked coordinates: 1894, 404
38 - Clicked coordinates: 1806, 416
39 - Clicked coordinates: 22, 572

```

Diese Werte können nun in die initPointPairsXXX-Methoden übernommen werden. So dauert die Konfiguration einer neuen Kamera weniger als 15 Minuten.

#### 9.1.10.4 VideoCreator

Der VideoCreator ist eine Erweiterung, welche aufgrund von Performanceproblemen erstellt wurde. Sie ermöglicht es, aus den vom Hauptprogramm ausgegebenen Daten ein Video zu erstellen. Die grundsätzliche Funktionsweise wurde in diesem Dokument bereits erklärt. An dieser Stelle beschränken sich die Informationen also auf die grundsätzliche Bedienung.

Das Exportieren der Bilddaten ist relativ simpel. In der Klasse MultiTracker befinden sich folgende auskommentierte Zeilen, welche aktiviert werden:

```

/*
// Debug: Create output for demo video
imwrite("../resources/tools/videocreator/input/" + std::to_string(frameId) + "_frameHud.jpg", frameHud);
imwrite("../resources/tools/videocreator/input/" + std::to_string(frameId) + "_frameMar.jpg", frameMar);
imwrite("../resources/tools/videocreator/input/" + std::to_string(frameId) + "_frameMic.jpg", frameMic);
*/

```

Im TrackingModule befinden sich folgende Zeilen an zwei Stellen. An beiden Orten werden die Zeilen aktiviert.

```

/*
// Debug: Create output for demo video
imwrite("../resources/tools/videocreator/input/" + std::to_string(frameId) + "_tracking.jpg", fieldModel);
*/

```

Ebenfalls im TrackingModule können folgende Zeilen aktiviert werden, um die Distanzen auszugeben:

```
/*
// Debug: Create distance output for videocreator
int distance = sqrt(
    pow((double) (histPlayer.getPositionInModel().x - nearestPlayer.getPositionInModel().x), 2.0) + pow((double)
    (histPlayer.getPositionInModel().y - nearestPlayer.getPositionInModel().y), 2.0));
cout << "DISTANCE-distances" << histPlayer.getCamerasPlayerId() << ".insert(std::make_pair(" << frameld << ", << distance << ));" <<
endl;
*/
```

Wurden diese Änderungen vorgenommen, kann die Applikation MultiTracker wie gewohnt gebaut und ausgeführt werden. Sobald die Applikation fertig ist, wurden alle nötigen Bilder in den Ordner «resources/tools/videocreator/input/» kopiert. Die Distanzen wiederum können auf der Kommandozeile gefunden werden. Am besten wird der gesamte Output in ein Excel kopiert und dort nach «DISTANCE» gefiltert. Anschliessend kann mit dem Ersetz-Tool «DISTANCE» durch «» ersetzt werden. Danach beinhaltet das Excel Blatt ausschliesslich Zeilen, die verwendet werden können. Alle diese Zeilen können nun im Tool VideoCreator in die Methode «init» kopiert werden.

```
void init() {
    distances1.insert(std::make_pair(2, 4));
    distances102.insert(std::make_pair(2, 2));
    distances103.insert(std::make_pair(2, 14));
    ...
}
```

Anschliessend kann der VideoCreator ausgeführt werden und es wird aus den echt berechneten Daten des MultiTrackers ein Video erstellt.

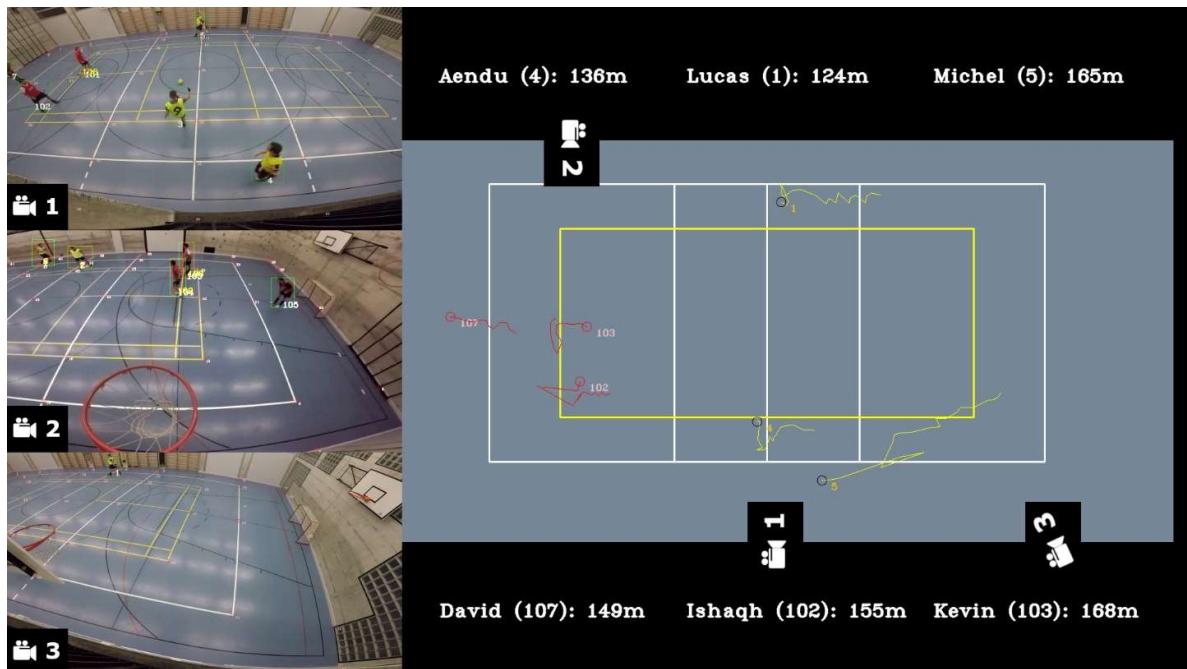


Abbildung 57 Ein Frame des aus dem VideoCreator entstehenden Videos für die Testsequenz.

## 9.2 Abbildungsverzeichnis

Abbildung 1 Beispiele für die Videodaten.	9
Abbildung 2 Beispiel für die eigenen Videodaten.	9
Abbildung 3 Beispiel für die verbesserten eigenen Videodaten.	10
Abbildung 4: Grober Ablauf für die Verarbeitung der Bilder dreier Kameras.	11
Abbildung 5: Ein Beispiel dreier ausgelesener Bilder (eines pro Kamera zum selben Zeitpunkt).	12
Abbildung 6 Beispiel einer Personendetektion für ein Kamerabild.	12
Abbildung 7 Vergleich der Detektion für ein Frame. YOLOv2 (links) erkennt deutlich weniger Spieler als YOLOv3 (rechts).	13
Abbildung 8 Links ist ein Teil eines Kamerabilds mit allen Spielern zu sehen. Rechts sind die sechs daraus gewonnenen Spielerbilder ersichtlich.	15
Abbildung 9 Ein gelber Spieler und die Resultate nach den Thresholds X (Mitte) und Y (rechts). Es wird nicht viel als rot erkannt.	16
Abbildung 10 Ein roter Spieler und die Resultate nach den Thresholds X (Mitte) und Y (rechts). Im Gegensatz zu einem gelben Spieler überleben genug Pixel die Thresholds, sodass dieser Spieler als «rot» erkannt wird.	16
Abbildung 11 Beispiele der verwendeten T-Shirt-Fotos.	16
Abbildung 12 Beispielbilder aus dem Trainingsset. Alle sechs Klassen sind enthalten: "Aendu", "Daevu", "Michel", "Lucas", "Ishaqh" und "Kevin".	17
Abbildung 13 Der beste Durchlauf mit den Verläufen für Accuracy (oben, blau) und Loss (unten, rot).	18
Abbildung 14 Acht Beispielbilder, welche korrekt klassifiziert werden (links), sowie die zwei Falschklassifikationen (rechts).	19
Abbildung 15 Der rote Punkt (in Pixelkoordinaten) wird für die Positionsumrechnung verwendet.	20
Abbildung 16 Plan der im Projekt verwendeten Halle. Er wurde massstabsgerecht in Modellkoordinaten erfasst. Auch sichtbar sind einige Referenzpunkte.	20
Abbildung 17 Referenzpunkte in einem Perspektiven-Kamera-Bild.	21
Abbildung 18 Ein Spieler steht im Perspektivensystem einer Kamera. Es müssen drei Referenzpunkte ausgewählt werden.	22
Abbildung 19 Es wurden drei Referenzpunkte für die Spielerposition ausgewählt. Die Position wird nun anhand dieser Referenzpunkte in baryzentrischen Koordinaten ( $u, v, w$ ) ausgedrückt.	22
Abbildung 20 Die baryzentrischen Koordinaten ( $u, v$ und $w$ ) aus der letzten Abbildung werden im Modellsystem verwendet, um die Spielerposition anzunähern.	22
Abbildung 21 Ein Ausschnitt eines Perspektivenbildes einer Kamera mit den darin erkannten Spielern.	23
Abbildung 22 Die errechneten Positionen der Spieler in der letzten Grafik, dargestellt im Modellsystem inkl. der zur Berechnung verwendeten Referenzpunkte (gelbe Linien).	23
Abbildung 23 Ein Teil der Dreiecke ist hier in schwarz dargestellt. Im vollständig konfigurierten Programm ist das ganze Feld in Dreiecke aufgeteilt.	24
Abbildung 24 Die gleichen Dreiecke der letzten Abbildung nun in der Perspektive.	25
Abbildung 25 Die dem Spieler "5" nächsten Referenzpunkte wurden rot mit ihm verbunden.	25
Abbildung 26 Die drei Kamerasichten auf das Geschehen zu einem zufälligen Zeitpunkt.	27
Abbildung 27 Der aus den in der letzten Abbildung gezeigten Kamerasichten entstandene Input für das Tracking-Modul.	28
Abbildung 28 Das erste Frame der Kamera "cameraHud" im Testsegment.	28
Abbildung 29 Beispielbild für das Wiederfinden bestehender Spieler.	29
Abbildung 30 Ein Frame des Demonstrationsvideos gegen Ende der bearbeiteten Sequenz.	30
Abbildung 31 Beispiele für die schlechtere Erkennung mit der kleineren Skalierung. Links: Doppelte Erkennungen. Rechts: Fehlende Erkennung.	34

Abbildung 32 Der gleiche Spieler zur vermeintlich exakt gleichen Zeit. Es ist ersichtlich, dass der Spieler im rechten Bild schon losprintet, während er im linken Bild noch beobachtet.	40
Abbildung 33 Beispiel eines Synchronisationsfehlers.	40
Abbildung 34 Links: Ein unerkannter Spieler. Mitte: Doppelt erkannter Spieler. Rechts: Sonstige Fehlererkennung, welche «Fusion» getauft wurde.	41
Abbildung 35 Die gelbe Linie stellt die Fehlerdistanz zwischen der effektiven Spielerposition (blauer Punkt) und der vom Algorithmus verwendeten Position (roter Punkt) dar.	41
Abbildung 36 Obwohl die korrekte Position des Spielers beim roten Punkt läge, wird die berechnete Position (gelber Punkt) als korrekt bewertet, da die Bounding Box relevant ist, nicht die Spielerposition.	43
Abbildung 37 Frame 501 der Testsequenz.	45
Abbildung 38 Frame 502 der Testsequenz.	46
Abbildung 39 Frame 503 der Testsequenz.	47
Abbildung 40 Frame 504 der Testsequenz.	48
Abbildung 41 Frame 507 der Testsequenz.	49
Abbildung 42 Frame 521 der Testsequenz.	50
Abbildung 43 Frame 523 der Testsequenz.	51
Abbildung 44 Frame 527 der Testsequenz.	52
Abbildung 45 Frame 531 der Testsequenz.	53
Abbildung 46 Frame 561 der Testsequenz.	54
Abbildung 47 Zeigt die Sicht der cameraHud zum aktuellen Zeitpunkt sowie die erkannten Spieler inklusive der für diese Kamera lokalen Spieler-IDs.	63
Abbildung 48 Zeigt die Sicht der cameraMar zum aktuellen Zeitpunkt sowie die erkannten Spieler inklusive der für diese Kamera lokalen Spieler-IDs. Hier ebenfalls ein Beispiel eines als Duplikat erkannten Spielers (ID 105).	63
Abbildung 49 Zeigt die Sicht der cameraMic zum aktuellen Zeitpunkt sowie die erkannten Spieler inklusive der für diese Kamera lokalen Spieler-IDs. Hier ebenfalls ein Beispiel eines als Duplikat erkannten Spielers (Spieler mit der Trikotnummer 2).	64
Abbildung 50 Die Kamerasicht von cameraHud übertragen auf das Modell des Feldes. Die Punkte stellen die Spieler dar, die lokale ID des Spielers wird neben dem Punkt angezeigt. Zudem zeigen die gelben Linien für jeden Spieler, welche Referenzpunkte verwendet wurden.	64
Abbildung 51 Die Kamerasicht von cameraMar übertragen auf das Modell des Feldes. Die Punkte stellen die Spieler dar, die lokale ID des Spielers wird neben dem Punkt angezeigt. Zudem zeigen die gelben Linien für jeden Spieler, welche Referenzpunkte verwendet wurden.	65
Abbildung 52 Die Kamerasicht von cameraMic übertragen auf das Modell des Feldes. Die Punkte stellen die Spieler dar, die lokale ID des Spielers wird neben dem Punkt angezeigt. Zudem zeigen die gelben Linien für jeden Spieler, welche Referenzpunkte verwendet wurden.	65
Abbildung 53 Der Input für den Tracking-Algorithmus. Dieser besteht aus allen von den drei Kameras erkannten Punkten in Modell-Koordinaten. Nummern zwischen 1000 und 2000 gehören zu cameraMic, Punkte zwischen 2000 und 3000 kommen von cameraHud, die andern von cameraMar.	66
Abbildung 54 Das Trackingresultat, welches die Spieler (rote und schwarze Punkte), deren vergangenen Positionen (gelbe Linien) sowie - falls konfiguriert - die Groundtruth (grüne Punkte) anzeigt.	66
Abbildung 55 Für jeden Klick in das Tracking-Fenster erscheint eine solche Abfrage im Konsolenfenster.	67
Abbildung 56 Ein Beispiel der Erfassung einiger Referenzpunkte.	68
Abbildung 57 Ein Frame des aus dem VideoCreator entstehenden Videos für die Testsequenz.	70

### 9.3 Tabellenverzeichnis

Tabelle 1 An dem Projekt beteiligte Personen und ihre Rollen.	5
Tabelle 2 Zeitplan und Meilensteine.	8
Tabelle 3 Aufnahmefotos Magglingen.	9
Tabelle 4 Aufnahmefotos der ersten eigenen Aufnahmen.	10
Tabelle 5 Aufnahmefotos der zweiten eigenen Aufnahmen.	10
Tabelle 6 Zusammensetzung des Trainings-Datensets.	17
Tabelle 7 Zusammensetzung des Validierungs-Datensets.	18
Tabelle 8 Spielerindex für nebenstehende Abbildung.	27
Tabelle 9 Performancemessung für fünf Durchläufe.	33
Tabelle 10 Gesamtdauervergleich eines Durchlaufs anhand von Backend und Target.	35
Tabelle 11 Wichtige Entscheidungen, die während des Projekts getroffen wurden.	39
Tabelle 12 Fehleranalyse beim Auslesen der Bilder.	40
Tabelle 13 Fehleranalyse der Personenerkennung mit YOLO.	41
Tabelle 14 Fehleranalyse Bounding Box.	42
Tabelle 15 Fehleranalyse bei der Farberkennung.	42
Tabelle 16 Fehleranalyse bei der Positions berechnung.	43

### 9.4 Glossar

#### **Accuracy**

Prozentsatz, welcher für eine Menge von Inputs die Korrektheit der Outputs darstellt.

#### **BLOB**

Abkürzung für "Binary Large Object", also ein grosses binäres Datenobjekt.

#### **Boolean**

Datentyp, entspricht entweder dem Wert "wahr" oder "falsch".

#### **Bounding Box**

Rahmen um ein Objekt, in diesem Projekt ist der Rahmen, welcher die Spieler umgibt, gemeint.

#### **Branch**

Ein eigener Zweig im GIT, in welchem Änderungen unabhängig von anderen Branches verwaltet werden können.

#### **C++**

Programmiersprache.

#### **Classifier**

Wird verwendet, um Objekte nach Eigenschaften zu klassifizieren.

#### **cmake**

Ein Werkzeug zur Erstellung von Projekten.

#### **Feature Point**

Ein spezieller Punkt in einem Bild oder aus einem Bild generiert, welcher zur Beschreibung des Bildes dienen kann.

#### **FPS**

Frames per second, sagt aus, wie viele Bilder pro Sekunde gezeigt oder verarbeitet werden.

#### **Frame**

Ein Einzelbild eines Films. Im Umkehrschluss ist ein Film eine Aneinanderreihung von Frames.

#### **Full HD**

Eine Bildauflösung von 1920 x 1080 Pixeln.

#### **Gewichte**

Gewichtswerte eines Neuronalen Netzes.

#### **Git**

Ein Tool zur Versionsverwaltung von Dateien, meistens verwendet beim Arbeiten mit Programmiercode.

<b>Github</b>	Eine Webseite, auf welcher Git-Projekte verwaltet und gespeichert werden können.
<b>Groundtruth</b>	Wahrheitswerte, welche mit dem effektiven Resultat des Algorithmus verglichen werden können.
<b>History</b>	Als History (Geschichte) wird in diesem Projekt die Datenstruktur zur Speicherung der in der Vergangenheit erkannten Spieler bezeichnet.
<b>HSV</b>	Abkürzung für "Hue, saturation and value". Farbmodell basierend auf Farbwert, Sättigung und Dunkelstufe.
<b>Klonen</b>	Erstellen eines Abbildes eines Git-Repositories auf der eigenen Maschine.
<b>KW</b>	Steht für Kalenderwoche.
<b>Label</b>	Von einem Neuronalen Netz erkannte Klasse eines Inputs.
<b>Library</b>	Eine Programmbibliothek, welche Klassen und Funktionen für einen spezifischen Zweck anbieten.
<b>Loss</b>	Wert, welcher für ein Neuronales Netz ausdrückt, wie gut das Netz für ein Datenset funktioniert.
<b>Matlab</b>	Eine Applikation, welche die praxisnahe Arbeit mit mathematischen und wissenschaftlichen Themen ermöglicht.
<b>Mergen</b>	Das Zusammenführen von Branches im Git.
<b>Neuronales Netz</b>	Spezielle mathematische Konzepte für die Lösung eines Optimierungsproblems.
<b>OpenCV</b>	Steht für Open Source Computer Vision. Eine Library für die Umsetzung von Computer Vision.
<b>Operator</b>	Eine Person, welche für die Korrekturen des laufenden Trackings verantwortlich ist.
<b>Out-of-bounds</b>	Ein Zugriff ausserhalb eines Datenobjektes (Array, Liste).
<b>Perspektivenbild</b>	Bezeichnung für ein Kamerabild innerhalb dieses Projektes. Wird für die Unterscheidung zwischen Perspektivensystem und Modellsystem explizit so genannt.
<b>PointPair</b>	Eine Klasse, welche eine Zahl und zwei Punkte kapselt.
<b>Python</b>	Programmiersprache.
<b>Referenzpunkt</b>	Ein Punkt, welcher im Modell des Spielfeldes exakt definiert ist.
<b>Repository</b>	Ein Projekt im Git, kann als Datenstruktur angesehen werden.
<b>SIFT</b>	Steht für Scale-invariant feature transform. Eine Methode zur Beschreibung von Feature Points in Bildern.
<b>struct</b>	Einfache Datenstruktur im C++.
<b>Terminal</b>	Kommandozeilenapplikation im Ubuntu zur Ausführung von Befehlen.
<b>Tesseract</b>	Eine Software zur Erkennung von Text.
<b>Threshold</b>	

Eine Schranke beziehungsweise ein Schwellwert.

---

**Tool**

Eine hilfreiche Applikation; im Rahmen dieser Arbeit meistens verwendet als "Programmierhilfsmittel".

---

**Tracking**

In diesem Projekt: Die mit einer Applikation erreichte Verfolgung von sich bewegenden Personen.

---

**Visual Studio**

Eine Entwicklungsumgebung für verschiedene Programmiersprachen

---

**YOLO**

Ein System zur Detektion von Objekten. Wird in diesem Projekt für die Personenerkennung verwendet.

---

**ZIP**

Ein Dateiformat für komprimierte Dateien.

---

## 9.5 Quellenverzeichnis

- [1] «Warum sind technische Hilfsmittel so umstritten?». [Online] Verfügbar unter:  
<https://www.tagesspiegel.de/sport/fussball-wm2010/fehlentscheidungen-warum-sind-technische-hilfsmittel-so-umstritten/1871106.html>. [Zugegriffen: 29.12.2018]
- [2] «Moderner Daten-Fussball». [Online] Verfügbar unter:  
<https://www.aargauerzeitung.ch/sport/fussball/moderner-daten-fussball-was-man-mit-dem-sport-bh-alles-herausfinden-kann-130315760>. [Zugegriffen: 29.12.2018]
- [3] «12,7 Kilometer in 90 Minuten». [Online] Verfügbar unter:  
[https://www.nzz.ch/127\\_kilometer\\_in\\_90\\_minuten-1.763527](https://www.nzz.ch/127_kilometer_in_90_minuten-1.763527). [Zugegriffen: 08.12.2018]
- [4] «Digitec - GoPro Hero 5 Black». [Online] Verfügbar unter:  
<https://www.digitec.ch/de/s1/product/gopro-hero-5-black-12mp-30p-grau-actionkamera-5902028>.  
[Zugegriffen: 08.12.2018]
- [5] «OpenCV - VideoCaptureProperties». [Online] Verfügbar unter:  
[https://docs.opencv.org/3.4.2/d8/dfe/classcv\\_1\\_1VideoCapture.html#aa6480e6972ef4c00d74814ec841a2939](https://docs.opencv.org/3.4.2/d8/dfe/classcv_1_1VideoCapture.html#aa6480e6972ef4c00d74814ec841a2939). [Zugegriffen: 29.12.2018]
- [6] «OpenCV - Deep Neural Network Module». [Online] Verfügbar unter:  
[https://docs.opencv.org/3.4.2/d6/d0f/group\\_\\_dnn.html](https://docs.opencv.org/3.4.2/d6/d0f/group__dnn.html). [Zugegriffen: 29.12.2018]
- [7] «Github - ObjectDetectionSample». [Online] Verfügbar unter:  
[https://github.com/opencv/opencv/blob/3.4/samples/dnn/object\\_detection.cpp](https://github.com/opencv/opencv/blob/3.4/samples/dnn/object_detection.cpp). [Zugegriffen: 29.12.2018]
- [8] «Darknet - YOLO». [Online] Verfügbar unter:  
<https://pjreddie.com/darknet/yolo/>. [Zugegriffen: 29.12.2018]
- [9] J. Redmon, A. Farhadi. YOLOv3: An Incremental Improvement [Online] Verfügbar unter:  
<https://pjreddie.com/media/files/papers/YOLOv3.pdf>. [Zugegriffen: 29.12.2018]
- [10] «What's new in YOLO v3?». [Online] Verfügbar unter:  
<https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b>. [Zugegriffen: 29.12.2018]
- [11] «Github - YOLOv3 in Opencv». [Online] Verfügbar unter:  
<https://github.com/opencv/opencv/issues/11310>. [Zugegriffen: 22.12.2018]
- [12] «Opencv Answers - Detecting laser dots». [Online] Verfügbar unter:  
<http://answers.opencv.org/question/201692/detecting-laser-dots-and-get-the-coordinates/>.  
[Zugegriffen: 29.12.2018]
- [13] C. Ericson. Real-Time Collision Detection. 2015.
- [14] «Digitec - Lenovo Yoga 720-15». [Online] Verfügbar unter:  
<https://www.digitec.ch/de/s1/product/lenovo-yoga-720-15-1560-full-hd-intel-core-i7-7700hq-16gb-ssd-notebook-6392525>. [Zugegriffen: 29.12.2018]
- [15] «OpenCV - Net». [Online] Verfügbar unter:  
[https://docs.opencv.org/3.4.2/db/d30/classcv\\_1\\_1dnn\\_1\\_1Net.html](https://docs.opencv.org/3.4.2/db/d30/classcv_1_1dnn_1_1Net.html). [Zugegriffen: 29.12.2018]
- [16] «Github - DNN Efficiency». [Online] Verfügbar unter:  
<https://github.com/opencv/opencv/wiki/DNN-Efficiency>. [Zugegriffen: 25.12.2018]
- [17] «Github - Intel's Deep Learning Inference Engine backend». [Online] Verfügbar unter:  
<https://github.com/opencv/opencv/wiki/Intel%27s-Deep-Learning-Inference-Engine-backend>.  
[Zugegriffen: 29.12.2018]
- [18] «Github - DLDT». [Online] Verfügbar unter:

- <https://github.com/opencv/dldt>. [Zugegriffen: 25.12.2018]
- [19] «Gitflow Workflow». [Online] Verfügbar unter: <https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow>. [Zugegriffen: 15.12.2018]
- [20] «Install OpenCV 3 on Windows». [Online] Verfügbar unter: <https://www.learnopencv.com/install-opencv3-on-windows/>. [Zugegriffen: 29.12.2018]
- [21] «OpenCV – Installation on Linux». [Online] Verfügbar unter: [https://docs.opencv.org/3.4.2/d7/d9f/tutorial\\_linux\\_install.html](https://docs.opencv.org/3.4.2/d7/d9f/tutorial_linux_install.html). [Zugegriffen: 29.12.2018]
- [22] «Calculate on which side of a straight line is a given point located». [Online] Verfügbar unter: <https://math.stackexchange.com/questions/274712/calculate-on-which-side-of-a-straight-line-is-a-given-point-located>. [Zugegriffen: 30.12.2018]
- [23] «Halide». [Online] Verfügbar unter: <http://halide-lang.org/>. [Zugegriffen: 30.12.2018]
- [24] «Train Deep Learning Network to Classify New Images». [Online] Verfügbar unter: <https://ch.mathworks.com/help/deeplearning/examples/train-deep-learning-network-to-classify-new-images.html>. [Zugegriffen: 07.01.2019]
- [25] «Matlab - Googlenet». [Online] Verfügbar unter: <https://ch.mathworks.com/help/deeplearning/ref/googlenet.html>. [Zugegriffen: 07.01.2019]
- [26] «How do I export a Neural Network from MATLAB?». [Online] Verfügbar unter: <https://ch.mathworks.com/matlabcentral/answers/325291-how-do-i-export-a-neural-network-from-matlab>. [Zugegriffen: 07.01.2019]
- [27] «Importing models from ONNX to Caffe2». [Online] Verfügbar unter: <https://github.com/onnx/tutorials/blob/master/tutorials/OnnxCaffe2Import.ipynb>. [Zugegriffen: 07.01.2019]
- [28] «OpenCV - Load Caffe framework models». [Online] Verfügbar unter: [https://docs.opencv.org/3.4/d5/de7/tutorial\\_dnn\\_googlenet.html](https://docs.opencv.org/3.4/d5/de7/tutorial_dnn_googlenet.html). [Zugegriffen: 07.01.2019]
- [29] «CPU Performance Comparison of OpenCV and other Deep Learning frameworks». [Online] Verfügbar unter: <https://www.learnopencv.com/cpu-performance-comparison-of-opencv-and-other-deep-learning-frameworks/>. [Zugegriffen: 09.01.2019]
- [30] «Using OpenVINO with OpenCV». [Online] Verfügbar unter: <https://www.learnopencv.com/using-openvino-with-opencv/>. [Zugegriffen: 09.01.2019]
- [31] «Deep Learning based Object Detection using YOLOv3 with OpenCV». [Online] Verfügbar unter: <https://www.learnopencv.com/deep-learning-based-object-detection-using-yolov3-with-opencv-python-c/>. [Zugegriffen: 09.01.2019]
- [32] «Kick-Off Bachelor Thesis». [Online] Mit korrektem Login und Zugang zum Kurs verfügbar unter: [https://moodle.bfh.ch/pluginfile.php/908344/mod\\_resource/content/3/KickOffBachelorThesis.pdf](https://moodle.bfh.ch/pluginfile.php/908344/mod_resource/content/3/KickOffBachelorThesis.pdf). [Zugegriffen: 09.01.2019]

## 9.6 Arbeitsjournal

Da das Arbeitsjournal sehr ausführlich und dementsprechend lang ist, wurde es aus dem Bericht ausgelagert. Es ist online unter folgendem Link verfügbar:

[https://www.dropbox.com/s/7paew7vgopuj4cv/\\_Planung.xlsx?dl=0](https://www.dropbox.com/s/7paew7vgopuj4cv/_Planung.xlsx?dl=0)

Dieses Dokument beinhaltet auch die Meilensteine und einen kleinen Teil der ursprünglichen Konzeption. Das Arbeitsjournal ist im Raster «Planung» zu finden.

Sollte der Link in der Zukunft nicht mehr funktionieren, kann das Dokument bei der Projektleitung bezogen werden.

## 9.7 Code

Da der Code der Arbeit aus vielen verschiedenen Klassen besteht, wurde darauf verzichtet, diese direkt im Anhang der Arbeit anzufügen. Im öffentlichen GIT-Repository auf Github ist Struktur und Code lesbarer und die Seite beinhaltet auch eine vollständige Historie der Arbeiten in der Form von Commits. Der Code befindet sich hier: <https://github.com/SoullessStone/MultiPersonCameraTrackingPOC>

## 10 Versionskontrolle

Version	Datum	Beschreibung	Autor
0.1	16.11.2018	Initiales Dokument	Michel Utz
0.2	08.12.2018	Struktur und Grunddokumentation	Michel Utz
0.3	09.12.2018	Erweiterung der Grunddokumentation	Michel Utz
0.4	10.12.2018	Erweiterung der Grunddokumentation	Michel Utz
0.5	15.12.2018	Überarbeiten und Erweiterung	Michel Utz
0.6	22.12.2018	Fehler, Bilder einfügen	Michel Utz
1.0	28.12.2018	Letzte Anpassungen für erste vollständige Version	Michel Utz
1.1	29.12.2018	Logische Fehler analysieren	Michel Utz
1.2	30.12.2018	Tabellenverzeichnis, Diverses	Michel Utz
1.3	02.01.2019	Auslagern Arbeitsjournal, Erweiterung Dokumentation	Michel Utz
1.4	07.01.2019	Einfügen Dokumentation Transferlearning, kleinere Anpassungen	Michel Utz
1.5	09.01.2019	Erweiterung der YOLO-Performance-Diskussion, Korrekturen	Michel Utz
1.6	11.01.2019	Einarbeitung der Rückmeldung von Marcus Hudritsch	Michel Utz
1.7	14.01.2019	Korrekturen	Michel Utz
1.8	17.01.2019	Einarbeiten der hoffentlich letzten Korrekturen	Michel Utz

## 11 Selbstständigkeitserklärung

Ich bestätige mit meiner Unterschrift, dass ich meine vorliegende Bachelorthesis selbstständig durchgeführt habe. Alle Informationsquellen (Fachliteratur, Besprechungen mit Fachleuten, usw.) und anderen Hilfsmittel, die wesentlich zu meiner Arbeit beigetragen haben, sind in meinem Arbeitsbericht im Anhang vollständig aufgeführt. Sämtliche Inhalte, die nicht von mir stammen, sind mit dem genauen Hinweis auf ihre Quelle gekennzeichnet.

Name, Vorname

Utz, Michel

Datum

17.01.2019

Unterschrift

.....