Automatisierte Videoanalyse zur Generierung von Referenzdaten für Spiele in der RoboCup Standard Platform League

Masterarbeit

Alexander Stöwing

Matrikelnummer: 2787395

26. März 2018



Fachbereich Mathematik / Informatik
Studiengang Informatik

1. Gutachter: Dr. Tim Laue
2. Gutachter: Prof. Dr. Rolf Drechsler

Inhaltsverzeichnis

1 Einleitung **1**

1.1 Motivation 1

1.2 Ziele und Aufbau der Arbeit 3

2 Hintergrund **4**

2.1 RoboCup 4

2.2 B-Human 5

2.3 Der *NAO* 5

2.4 Die *GoPro* 6

3 Verwandte Arbeiten **8**

3.1 Arbeiten über Referenzsysteme 8

3.2 Arbeiten aus dem Bereich der Robotererkennung 11

3.3 Arbeiten aus dem Bereich der Kalibrierung 12

3.4 Arbeiten aus dem Bereich der Modellierung 13

4 Grundlagen **15**

4.1 Farbräume 15

4.1.1 RGB 15

4.1.2 HSV 15

4.2 Koordinatensysteme 17

4.2.1 Das Weltkoordinatensystem 17

4.2.2 Das Kamerakoordinatensystem 18

4.2.3 Das Bildkoordinatensystem 18

4.3 Die Lochkamera 19

4.4 Die Normalverteilung 20
4.4.1 Das *Gaussian Mixture Model* 21
4.5 Das Kalman-Filter 23
4.5.1 Das *Unscented Kalman-Filter* 25
4.6 Kantendetektion 27
4.6.1 Lineare Filter 27
4.6.2 Gradiententheorie auf Bildern 28
4.6.3 Der Sobel-Operator 30
4.7 Der FAST Algorithmus 31
4.8 Gruppentheorie zur Generierung von Zahlen 32
4.8.1 Definition einer Gruppe 32
4.8.2 Erweiterte Gruppendefinitionen 33
4.8.3 Beispiel 33

5 Beschreibung des Systems 35
5.1 Die verwendeten Bibliotheken 35
5.1.1 *OpenCV* 35
5.1.2 *Eigen* 35
5.1.3 *Qt* 36
5.2 Der Aufbau des Systems 36
5.2.1 Beschreibung der Pakete 36
5.2.2 Beschreibung der Klassen 38
5.3 Der Prozessaufbau 41
5.4 Diskussion 41

6 Der Fokus auf das Spiel 43
6.1 Motivation dieser Entwicklung 43
6.2 Diskussion verschiedener Ansätze 45
6.3 Farbkalibrierung über *Gaussian Mixture Model* 47
6.3.1 Dadurch entstandene Möglichkeiten 49

7 Kalibrierung der *GoPro* 53
7.1 Diskussion der Kalibrierung 53

7.2 Intrinsische Kamerakalibrierung 55
7.3 Extrinsische Kamerakalibrierung 59
7.4 Transformationen zwischen den Koordinatensystemen 62
7.4.1 Welt-Zu-Kamera 62
7.4.2 Kamera-Zu-Welt 63
7.5 Diskussion 64

8 Erfassung der Roboter **65**
8.1 Diskussion der Möglichkeiten 65
8.2 Initiale Erfassung der Roboter 66
8.3 Tracking bereits erfasster Roboter 71
8.3.1 Resultierende Änderungen in der Detektion 72
8.4 Betrachtung möglicher Probleme 73

9 Modellierung der Roboter **77**
9.1 Der Aufbau des UKF 77
9.2 Perzeptintegration 80
9.3 Weitere Annahmen und Verbesserungen 82
9.3.1 Diskussion des Aufbaus 83
9.4 Speichern der Referenzdaten 86

10 Evaluation **88**
10.1 Vorwort zu der Evaluation 88
10.2 Ermittlung des Detektionsfehlers 90
10.3 Evaluation der initialen Spielerkennung 94
10.3.1 1. Video: Indoor 1 96
10.3.2 2. Video: Indoor 2 97
10.3.3 3. Video: Outdoor 97
10.3.4 Gesamt 97
10.4 Evaluation des Groundtruth-Trackings 99
10.5 Evaluation der Farbkalibrierung 102
10.5.1 Konstante Farbspezifikation 103
10.5.2 Adaptive Farbspezifikation 103

10.6 Evaluation der Blickwinkelunabhängigkeit 104
10.6.1 Erster Aufbau 104
10.6.2 Zweiter Aufbau 105
10.6.3 Dritter Aufbau 106
10.6.4 Vierter Aufbau 107
10.6.5 Gesamtvergleich 108
10.7 Evaluation der Auswirkung von Verdeckung 109
10.7.1 Versuch 1 110
10.7.2 Versuch 2 111
10.8 Laufzeit-Betrachtungen 114
10.9 Test des Gesamtsystems 115

11 Fazit und Ausblick **119**
11.1 Auswertung 119
11.1.1 Der Detektionsfehler 119
11.1.2 Die initiale Spielererkennung 120
11.1.3 Das Groundtruthtracking 120
11.1.4 Die Farbkalibrierung 121
11.1.5 Die Blickwinkelunabhängigkeit 121
11.1.6 Die Auswirkung von Verdeckungen 122
11.1.7 Die Laufzeiten der Anwendung 122
11.1.8 Der Gesamttest 123
11.2 Ausblick 124
11.3 Fazit 125
11.4 Danksagung 126

Literatur **130**

Kapitel 1

Einleitung

Dies ist eine Arbeit, die im Rahmen des Teams B-Human entstand. B-Human ist ein Team der Universität Bremen in Kooperation mit dem Deutschen Forschungszentrum für künstliche Intelligenz, welches regelmäßig und erfolgreich an den internationalen Wettbewerben des RoboCups [The RoboCup Federation, 2016b] in der Standard Platform League (SPL) [The RoboCup Federation, 2016c] teilnimmt und dabei insgesamt sechsmal den Weltmeistertitel gewinnen konnte. In der SPL spielen *NAO*-Roboter (vgl. Kapitel 2.3) in 5-er Teams gegeneinander Fußball.
Wie viele erfolgreiche Teams dokumentiert B-Human die eigenen Spiele durch Videoaufnahmen. Die Kamera, die vom Team für diesen Zweck verwendet wird, ist eine *GoPro* (vgl. Kapitel 2.4). Diese Aufnahmen wurden bisher von Teammitgliedern für eine spätere Fehleranalyse betrachtet oder für die Pflege der sozialen Netzwerke verwendet. Nun sollen diese Videos eine computergestützte Analyse ermöglichen, deren Herausforderungen und Möglichkeiten im nachfolgendem Kapitel erläutert werden.

1.1 Motivation

Diese Abschlussarbeit behandelt die Entwicklung eines Systems zur Videoanalyse eines Roboterfußballspiels in der SPL. Dabei werden viele Themenbereiche behandelt, wie zum Beispiel Kalibrierung, Bildverarbeitung und Modellierung von Unsicherheit. Ein Teil der Motivation dieser Arbeit ist es, dem Team eine Softwarebasis bereitzustellen, mit dem Analysen auf gesammeltem Videomaterial ermöglicht werden, welche zu weiterem Erkenntnisgewinn führen. Die wissenschaftliche Motivation zielt darauf ab, in den Bereichen “Videoanalyse und Tracking“ und “Robotik und Verifikation“ Fortschritte zu verzeichnen, welche aus dem engen Rahmen dieser Arbeit abstrahiert werden können. Warum der Fortschritt in diesen Bereichen interessant ist, wird in den jeweiligen Unterkapiteln diskutiert.

Videoanalyse und Tracking

Der Bereich der Videoanalyse wird zunehmend interessanter, da viele Sicherheitssysteme auf Kameras basieren. Dadurch entstehen Unmengen an Material zur Sichtung, deren manuelle Analyse Kosten verursacht. Dabei ist eine computergestützte Analyse nicht nur von ökonomischem Interesse, sondern in vielen Bereichen auch eine Frage der Sicherheit. In manchen Anwendungsfeldern kann es dabei um Menschenleben gehen, bei denen Computersysteme Sicherheitspersonal helfen könnten, mit der Datenmasse umzugehen. Dies würde damit die menschliche Fehleranfälligkeit zumindest zum Teil ausgleichen, da der Mensch bei langen gleichbleibenden Sequenzen dazu neigt, unaufmerksam zu werden.
Eine andere Branche, die von einer solchen Entwicklung profitieren könnte, ist die Unterhaltungsindustrie. Viele Sportarten werden gefilmt und analysiert. Dazu beschäftigen sich manche Arbeiten mit einer computergestützten Highlight-Extraktion von Sport-Videos, wie in Ren und Jose [2009], welche über hervorstechende Merkmale den psychologischen Messwert der *Attention* in einem Bild berechnen. Ebenfalls könnte man die Spieler in dem Videomaterial verfolgen, ohne dass störende Sensorik an diesen befestigt werden müsste, wie es in Misu u. a. [2009] getan wurde. Ausgehend von diesem Ansatz wären viele weitere Nutzungen denkbar, wie zum Beispiel eine vollständige statistische Auswertung der Spiele zur Unterstützung des Trainings.
In dieser Arbeit wird eine automatisierte Videoanalyse zur Anwendung in der SPL untersucht. Sollten in diesem Kontext Erkenntnisse erzielt werden, so kann man diese auch in abstrahierter Form auf andere Tracking- und Videoanalyse-Anwendungen übertragen.

Robotik und Verifikation

In der Robotik wird mit komplizierten mechanischen Konstrukten und mindestens ebenso komplexer Software gearbeitet. Obwohl diese Kombination in vielen Bereichen durchaus Erfolg hat, ist sie fehleranfällig. Bei ausreichender Präzision des Trackings in dieser Abschlussarbeit kann man dann die generierten Daten als Referenzdaten verwenden. Diese sind bei vielen Entwicklungen in der Robotik nötig, um überhaupt eine Aussage über die Güte treffen zu können. Als Beispiel sei hier die Entwicklung von Nisticò u. a. [2007] genannt, in welcher der Ball von einer Gruppe von Robotern getrackt werden soll. In ihrer Beschreibung des Experiments wird erwähnt, dass sich die Suche nach einem geeignetem Referenzsystem für Positionen als sehr schwierig erwies [vgl. Nisticò u. a., 2007, Kapitel 4].
Die Anwendungsmöglichkeiten der produzierten Daten kann man nicht nur auf Verifikation in der Robotik beschränken, in der man die Daten mit Lokalisierungs- oder Perzeptionsergebnissen vergleicht. Die Daten können auch bei der Entwicklung neuer Anwendungen helfen. In der Robotik setzen viele Lernverfahren auf Referenzwerte und wollen zum Beispiel den Fehler zum tatsächlichen Wert ermitteln. Dabei ist die Beschaffung dieser „wahren“ Werte ein kritischer Punkt. Hier können die generierten Daten Abhilfe schaffen, mit denen man Trainingsdaten für Lernverfahren erstellen kann. Ein Beispiel hierfür wäre die Arbeit von

Raul Rojas und Förster [2006], bei der ein Bewegungsmodell eines Roboters gelernt wird. Um dort eine Korrektur des Modells vorzunehmen, wird eine Kamera benutzt, um die exakte Position des Roboters zu ermitteln [vgl. Raul Rojas und Förster, 2006, Kapitel 4.2].
Das Resultat präziser Referenzdaten wären zuverlässige Roboter, welche in einer Gesellschaft, die immer mehr Roboter im Alltag nutzt, äußerst wünschenswert sind.

1.2 Ziele und Aufbau der Arbeit

Das Ziel dieser Arbeit ist es, ein Trackingsystem zu entwickeln, welches die Spiele in der SPL im Nachhinein auswertet. Dabei soll das System Videoaufnahmen einer Kamera (*GoPro*) analysieren und die Positionen von NAO-Robotern möglichst genau extrahieren, ohne dabei nach Regeln unzulässige Markierungen zu nutzen. Zusätzlich soll sich das System gegenüber den Herausforderungen der SPL als robust erweisen.
Um den Kontext dieser Arbeit zu verdeutlichen, folgt mit Kapitel 2 der Hintergrund. Das System vereint viele Themengebiete, die für eine erfolgreiche Bearbeitung beachtet und diskutiert werden müssen, weshalb anschließend die verwandten Arbeiten in Kapitel 3 einen Einblick in die verschiedenen Bereiche bieten. Die absoluten Grundlagen zum Verständnis dieser Arbeit werden in Kapitel 4 behandelt, damit das Grundvokabular für die nachfolgenden Kapitel vorliegt. Im Anschluss wird eine Beschreibung des Systems in Kapitel 5 gegeben, damit eine Idee der Umsetzung vermittelt wird. Nach dieser Beschreibung wird der Ablauf des eigentlichen Trackings behandelt. Dies startet mit dem Fokus in Kapitel 6, in dem allgemeine Probleme des Trackings und der Lösungen behandelt werden. Danach wird die Kalibrierung in Kapitel 7 angesprochen, welche für die Bildverarbeitung und Generierung der Daten essenziell ist. Hierauf folgt die Robotererkennung in Kapitel 8, welche den Bildverarbeitungsalgorithmus zur Erfassung der Roboter beschreibt. Die Verarbeitung dieser Daten zu Referenzdaten wird dann in Kapitel 9 beschrieben.
Nach der vollständigen Beschreibung des entwickelten Systems wird dieses in Kapitel 10 im Hinblick der hier gesetzten Ziele evaluiert. Danach wird in Kapitel 11 ein Fazit zu dieser Arbeit gezogen und ein Ausblick auf mögliche zukünftige Entwicklungen gegeben.

Kapitel 2

Hintergrund

In diesem Kapitel wird ein kurzer Überblick über den Kontext der Arbeit gegeben, um dem Leser den RoboCup, B-Human und die verwendete Kamera nahezubringen.

2.1 RoboCup

Die „Robot World Cup Initiative“, auch „RoboCup“ genannt, ist eine internationale wissenschaftliche Initiative, um die Forschung im Bereich Künstliche Intelligenz und Robotik zu fördern. Dazu wurde das anspruchsvolle Langzeitziel definiert, Mitte des jetzigen Jahrhunderts mit einem Team vollkommen autonomer humanoider Roboter den FIFA Weltmeister auf dem Feld zu begegnen und zu besiegen [The RoboCup Federation, 2016b]. Dies galt im Gründungsjahr 1997 als ein sehr ehrgeiziges Ziel, welches den damals typischen Rahmen der statischen KI-Probleme sprengte und rege Zustimmung in der wissenschaftlichen Gemeinschaft fand. Seitdem finden jährlich Weltmeisterschaften und kleinere Wettbewerbe statt, um durch Austausch der Ergebnisse die Entwicklung voranzutreiben [The RoboCup Federation, 2016a]. Ein Ausschnitt ist auf Abb. 2.1 zu sehen, auf dem ein Spiel der Standard Platform League (SPL) stattfindet.

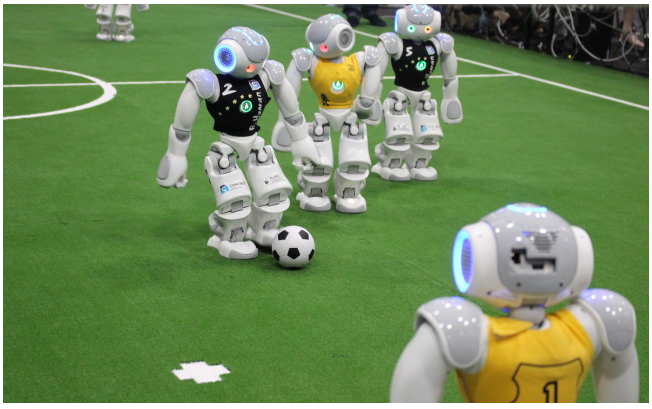


Abbildung 2.1 B-Human bei einem Spiel auf der Weltmeisterschaft 2017 in Nagoya (Japan).

Die Besonderheit der SPL innerhalb des RoboCups ist die alleinige Benutzung des *NAO*s als Plattform, welche auf Hardwareebene nicht modifiziert werden darf, weshalb es ein reiner Softwarewettbewerb ist [vgl. The RoboCup Federation, 2016c]. Die SPL ist ein Teil der Liga RoboCupSoccer, neben der weitere Ligen wie RoboCupRescue, RoboCup@Home, RoboCup-Industrial und RoboCupJunior existieren.

2.2 B-Human

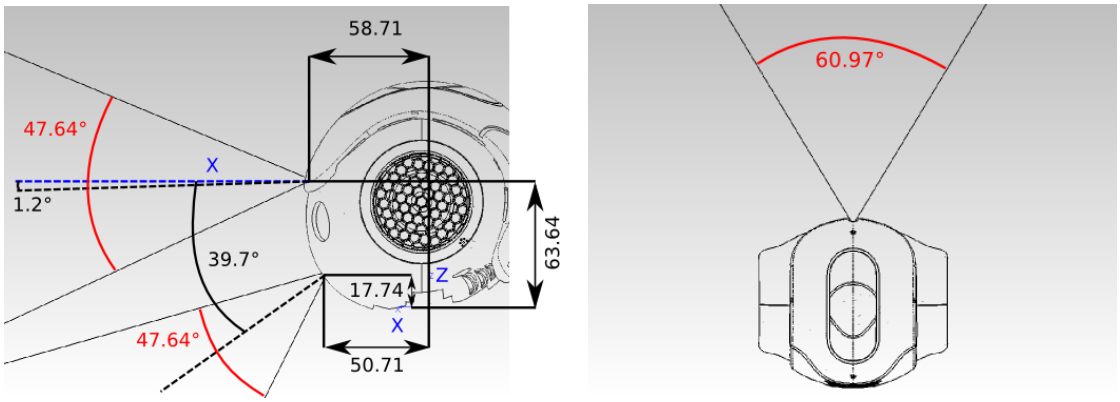
B-Human ist das RoboCup-Team der Universität Bremen in Kooperation mit dem Deutschen Forschungszentrum für künstliche Intelligenz, welches regelmäßig und erfolgreich an den Wettbewerben des RoboCups teilnimmt und jährlich einen Coderelease wie in Röfer u. a. [2017] veröffentlicht. Aktuell ist es eines der besten Teams der SPL und hatte in den Jahren 2009, 2010, 2011, 2013, 2016 und 2017 den Weltmeistertitel erringen können. Ergänzend dazu ist B-Human seit 2009 in der GermanOpen ungeschlagen [vgl. Team B-Human, 2015].

2.3 Der *NAO*

Der *NAO* von der Firma SoftBank Robotics, ehemals Aldebaran Robotics, ist ein humanoider Roboter, der in der aktuellen Version 57,4 cm groß und 5,4 kg schwer ist [SoftBank Robotics, o.D.(b)]. Dieser Roboter besitzt 25 steuerbare Freiheitsgrade für Bewegungen, eine Inertialeinheit zur Balancierung und Schätzung der Orientierung, etliche Drucksensoren zur Interaktion, zwei Lautsprecher für die Soundausgabe, vier Mikrofone für die Soundeingabe, je zwei Ultraschallsender und -empfänger zur Erfassung von Hindernissen und zwei Kameras zur Wahrnehmung der Umgebung. Der *NAO* ist ein programmierbarer Roboter und besitzt deswegen sowohl eine Ethernet- als auch eine USB-Schnittstelle und WIFI, um dem Nutzer die Verbindung zum Roboter zu ermöglichen [SoftBank Robotics, o.D.(a)]. Er besitzt außerdem eine Intel ATOM Z530 CPU mit 1,6 GHz sowie 1 GB RAM, 2 GB Flashspeicher und eine 8 GB Micro SDHC auf dem Motherboard [SoftBank Robotics, o.D.(c)]. Sein äußeres Erscheinungsbild ist auf Abb. 2.2 zu sehen.
Diese Ausstattung ermöglicht es den Teams in der SPL, eine Mannschaft von autonomen Robotern Fußball spielen zu lassen. Während der Bewegungsrahmen des Roboters voll ausgeschöpft wird, wobei die Beine durch einen gemeinsamen Motor in der Hüfte gesteuert werden, beschränken sich die meisten Teams auf die visuelle Wahrnehmung des Roboters, dessen Kameras eine Auflösung von bis zu 1280×960 Pixeln erreichen können. Diese befinden sich auf der Stirn und in Höhe des Mundes, wie in Abb. 2.3 dargestellt.



Abbildung 2.2 Der *NAO*, entnommen aus [SoftBank Robotics, o.D.(a)].



(a) Seitenansicht (b) Ansicht von Oben

Abbildung 2.3 Die Kameras des *NAO* mit eingezeichneter Blickrichtung samt Öffnungswinkel, entnommen aus [SoftBank Robotics, o.D.(d)].

2.4 Die *GoPro*

Auf dem Markt gibt es eine reiche Auswahl an Kameras, mit denen man hochwertige Bilder erhalten kann. Diese Arbeit wurde primär auf Basis einer bestimmten Kamera entwickelt und auf dieser getestet. Das Team B-Human hat eine *GoPro* Hero 3+ Black Edition in ihrer Ausrüstung, welche sich durch ein weites Sichtfeld auszeichnet. Diese Kamera, in Abb. 2.4 skizzenhaft dargestellt, ist ein recht kostengünstiges Modell, welches in der Lage ist, die Szenerie eines SPL-Spieles nahezu vollständig einzufangen.



Abbildung 2.4 Eine Skizzendarstellung der *GoPro*, entnommen aus [GoPro, 2017].

Dabei reichen die verfügbaren Auflösungen von 848×480 bis 4096×2160 [vgl. GoPro, 2017, S. 21]. Eine Aufnahme der Kamera von einer Spielszene ist in Abb. 2.5 zu sehen.



Abbildung 2.5 Eine Beispielaufnahme von einer *GoPro* mit einer Auflösung von 1920×1080.

Kapitel 3

Verwandte Arbeiten

Die hier vorgestellten verwandten Arbeiten sind in drei Kategorien eingeteilt. Die erste Kategorie behandelt Arbeiten über Referenzsysteme, bei denen die meisten Berührungspunkte mit dieser Arbeit bestehen, da in jeder Perzeption mit einem anderen Gebiet wie Kalibrierung und/oder Modellierung kombiniert wird. Anschließend werden die einzelnen Kategorien wie Erkennung von Robotern, Kalibrierung optischer Systeme und Modellierung im Sinne von Filterung der Perzepte behandelt.

3.1 Arbeiten über Referenzsysteme

Sheh und West [2005]: Entwickelten ein System, in dem ein Sony AIBO Roboter zur Erweiterung eines Smart-Home-Systems benutzt wird. Dieser soll Aufgaben und Bereiche wahrnehmen, die durch das integrierte System im Haus nicht abgedeckt werden können. Um dies zu tun, muss die Lokalisierung des Roboters korrekt sein. Dabei kann man die lokale Positionsschätzung des Roboters mit dem globalen Kamerasystem erweitern. Der Roboter liefert über eine farbbasierte Detektion eine relative Position zu einer Landmarke. Das globale System detektiert den Roboter über *Background-Subtraction* und wendet dann *Matching* an. Diese Messungen werden dann im Partikelfilter für die *Monte Carlo Lokalisierung* kombiniert.
Das System benutzt einen *Background-Subtractor*, der sich in der Entwicklung dieser Arbeit als schlecht anwendbar erwies. Interessant ist vor allem die Positionsdatengenerierung über eine Relation zu Landmarken, was im Kontext dieser Arbeit nicht möglich ist, aber in ähnlicher Form durch eine Kalibrierung gegeben ist.

Ruiz-del-Solar, Loncomilla und Vallejos [2007]: Arbeiteten an einem System für einen automatisierten Computerschiedsrichter, unter Anderem zur Analyse von Roboterfußballspielen. Dabei profitiert das System von der Erfassung mehrerer Aspekte des Spiels, wie zum Beispiel der Tore, der Landmarken und des Balles. Diese Objekte werden über Farbseg-

mentierung und Blobdetektion in Kombination mit einfachen Regeln geformt. Die Feldlinien werden über Kantendetektion und Houghtransformation erkannt und über die Landmarken identifiziert. Alle nicht detektierten, nicht grünen Objekte auf dem Feld sind dann Roboterkandidaten und werden mit SIFT-Features validiert und identifiziert. Eine Annahme, die das System nicht trifft, ist die der statischen Position. Die Lokalisierung der Kamera findet über Landmarken und intrinsische Kameraparameter statt, indem bewährte RoboCup Methoden verwendet werden. Zum Tracking der Objekte wird der Mean-Shift-Algorithmus verwendet, dessen Featuremodelle über ein Kalman-Filter geschätzt werden. Über das Tracking erhält man Informationen zu Bewegungen und Aktionen, die man für eine reduzierte RoboCup-Regelanwendung verwenden kann. Die Regeln, die in der Arbeit umgesetzt wurden sind Torgebung, Überprüfung der gültigen Kickoffpositionen, Aus des Balles, Global Game Stuck, Verlassen des Feldes durch einen Roboters und illegale Verteidigung.
Ruiz-del-Solar, Loncomilla und Vallejos [2007] haben ein System entwickelt, bei dem weiter gedacht wurde als für die Ziele dieser Arbeit nötig wäre, aber dennoch sind viele grundlegende Prinzipien gleich. Die Detektion basierend auf Farben und Features ist hier ebenfalls gegeben, wobei in vorherigen Versuchen die Formdetektion verworfen wurde.

Khandelwal und Stone [2012]: Nutzten zwei Microsoft Kinect RGB-D Sensoren, um Groundtruth Daten von *NAO*s in der SPL zu ermitteln. Dabei sollte die gesamte Erfassung ohne Marker auskommen, welche verstärkt in der Small Size League Bildverarbeitung [Zickler u. a., 2010] benutzt werden. Über ROS und PCL ist ein Framework gegeben, um die Daten der Kinect auszuwerten, die Bildkoordinaten in Weltkoordinaten zu transformieren und über Bildbeispiele auch die Farben zu kalibrieren. Diese Kalibrierungen müssen manuell beim Aufbau stattfinden. Die Robotererkennung sucht nach Punktclustern in bestimmten Feldregionen, die eine Mindesthöhe und -größe haben. Das Zentrum der detektierten Cluster liefert dann die X- und Y-Koordinaten für die Groundtruth. Der Cluster wird nach einer der beiden Teamfarben durchsucht und dann einem Team zugeordnet. Die Balldetektion erlaubt mehr mögliche Regionen im Feld, ist aber auf eine andere Höhe von Clustern mit der Farbe Orange zugeschnitten. Beide Detektionen verzichten bei Mehrdeutigkeiten auf Ergebnisse, um mögliche False-Positives auszuschließen.
Diese Arbeit besitzt sehr viele Berührungspunkte zu dieser Entwicklung, aber nutzt den sensorischen Vorteil der Tiefenmessung aus, welche hier nicht zur Verfügung steht. Dennoch basiert die Entwicklung von Khandelwal und Stone [2012] ebenfalls stark auf Open-Source-Software.

Rodrigues, Cardoso, Vilas, Silva, Rodrigues, Belguinha und Gomes [2014]: Entwickelten eine Anwendung, um die Analyse von Fußballspielen zu automatisieren und zu verbessern. Mit Hilfe von strategischen Schaukarten sollen die tatsächlichen Spiele über beliebige Kameras erfasst und evaluiert werden. Es wird durch einen halbautomatisierten Prozess ein grüner Farbschwellwert ermittelt, mit dem ein binarisiertes Bild für Grün erstellt wird. In

diesem Bild werden dann schwarze Blobs detektiert und als Spielerkandidaten behandelt. Auf diesen wird Konturdetektion, *Dilatation*, *Erosion* und ein Mediumfilter angewendet, um diese herauszuarbeiten. Jede der Flächen sollte eine Konturkorrespondenz haben. Wenn mehrere Blobs nahe beieinander liegen, werden diese über eine vertikale Maske zusammengeführt. Wenn der Grünanteil innerhalb der Fläche einen Schwellwert unterschreitet, wird der Kandidat als Spieler klassifiziert. Analog wird der Ball über weiße Pixel als ovaler Blob detektiert. Jeder Spieler wird über einen repräsentativen gemittelten Farbwert (Hue) einem Team zugeordnet. Der Spieler, der am weitesten von beiden Farben entfernt ist, ist der Schiedsrichter. Die Spieler werden über die gespeicherte Aufstellung identifiziert und weiter getrackt. Das Tracking erfolgt über ein Matching von nahe gelegenen Spielerdetektionen, die mit einigen Heuristiken zum Spielverlauf erweitert werden. Die Positionen der Objekte werden über eine Homographie berechnet, welche zuvor über detektierte Linen ermittelt wurde.
Die Verarbeitung von beliebigen Videoquellen ist in diesem System nicht vorgesehen, aber die Detektionsbeschreibung kommt der hier vorgestellten Beschreibung recht nahe. Die Zuordnung von Spielern zu Teams wäre für künftige Entwicklungen ein sinnvolles Prinzip.

Misu, Matsui, Clippingdale, Fujii und Yagi [2009]: Benutzten ein System aus zwei verschiedenen Kameras und komplexer Modellierung, um die Spieler des japanischen Profifußballs zu tracken. Dabei gibt es eine fixierte Weitwinkelkamera für die allgemeine Szenenerfassung und eine vollständig steuerbare Kamera, um gezielt Spieler zu erfassen. Das Ziel des Systems ist es, die durch *Backgroundsubtraction* erhaltenen Silhouetten den verschiedenen Spielern zuzuordnen und die damit verbundenen Trajektorien entsprechend markieren zu können. Dabei wurden diverse Unsicherheiten durch Bayes Inferenz modelliert, sodass partielle Erkennung der Detektoren, wie Gesicht und Rückennummer eines Spielers, mit Wahrscheinlichkeitsmodellierung kompensiert werden können. Das Tracking einer Trajektorie wird über ein Kalman-Filter realisiert und es werden Faktoren zur Bestimmung von Mehrdeutigkeiten berechnet, die bei steigender Unsicherheit die Zuweisungsverteilung einer Gleichverteilung annähern lassen. Die „Spieler“ werden über Nähe zu Referenzfarben in die Kategorien Feldspieler, Torwart und Schiedsrichter unterteilt, wobei die ersten Beiden noch weiter in Links und Rechts unterschieden werden. Um aktiv die steuerbare Kamera zu nutzen, schaut das System 5 Sekunden in die Zukunft und berechnet, welcher Spieler künftig am stärksten verdeckt wird.
Misu u. a. [2009] verwenden eine wesentlich komplexere Detektion, die viel mehr auf die Identifizierung der jeweiligen Spieler zugeschnitten ist. Dies ist bei dem *NAO* nicht nötig, aber eine Detektion von Rückennummern mit Zuordnung von Teams kann bei künftigen Entwicklungen eine höhere Trackingqualität erzeugen.

Weigel [2015]: Entwickelte im Rahmen seiner Abschlussarbeit ein videobasiertes Trackingsystem für Hallenfußballspieler in einem Trainingsszenario, welches eine Analyse ihrer Qualitäten ermöglichen soll. Nach einer Vorder- und Hintergrundtrennung wird ein antrainierter

Sliding-Window-Detektor für die Spielerfindung verwendet, welche anschließend über einen angelernten *k-Nächste-Nachbarn-Klassifikator* identifiziert werden, um ein Tracking zu ermöglichen.
Bei sehr ähnlicher Zielstellung wurde hier das Tracking von Hallenfußballspielern ermöglicht, indem über den Detektor schon die Spieler identifiziert wurden, was bei Robotern mit gleichem Erscheinungsbild nicht in der Form anwendbar ist. In diesem Kontext wäre eine Detektion der Rückennummer wie von Misu u. a. [2009] zielführender.

3.2 Arbeiten aus dem Bereich der Robotererkennung

Kaufmann, Mayer, Kraetzschmar und Palm [2005]: Haben zwei Robotererkennungsalgorithmen auf Basis von zwei kombinierten mehrschichtigen Neuronalen Netzen entwickelt. Die Aufgabe der Roboterdetektion wird hier mehrstufig mit je zwei Möglichkeiten auf jeder Ebene realisiert.

1. Berechnen eines Region of Interest (ROI): Farbübergangsdetektion oder Blobdetektion.
2. Extraktion von Features (Merkmalen): Orientierungshistogramme oder einfache Features.
3. Klassifizierung: Neuronales Netz in beiden Fällen.
4. Finale Entscheidung: Über Schwellwertverfahren.

Die letztendliche Erkennungsrate liegt bei beiden Algorithmen bei über 90%.
Analog zu den Arbeiten, die versuchen, menschliche Spieler zu erkennen, wird hier maschinelles Lernen benutzt, um die Intraklassenvarianz des Zielobjekts zu handhaben. Dies ist bei *NAO*s nicht zwingend nötig, kann aber gute Detektionsraten liefern. Maschinelles Lernen wird vom Autor auf Grund mangelnder Erfahrung in dem Gebiet nicht verwendet.

Wilking und Röfer [2005]: Haben ein Verfahren entwickelt, um die Sony AIBOs zu detektieren und deren Pose (Distanz, Winkel und Orientierung) zu erfassen. Für die Detektion wird Segmentierung benutzt, da die Roboter Farbmarker besitzen. Auf diesen werden Farbregionen mit Konturen berechnet. Über diese Eigenschaften werden Attribute wie Anzahl der Ecken, Konvexität und die Anzahl der Winkelklassen zwischen verschiedenen Liniensegmenten berechnet. Für die Klassifizierung wird *Decision Tree Learning* mit den generierten Attributen benutzt, welche dann mit Symbolen versehen werden. In der Analyse werden dann die Symbole über Heuristiken zusammen geführt, um einen Roboter darzustellen. Über seine Symbolgruppe lassen sich dann Poseneigenschaften über Fläche und Ausrichtung berechnen. Dies ist ein anderer Ansatz im Vergleich zu Kaufmann u. a. [2005], der maschinelles Lernen

nutzt, aber diesen mit der Berechnung von Attributen von Kandidaten kombiniert. Die Berechnung und Verifikation von Attributen wird in dieser Arbeit in der Detektion aufgegriffen.

3.3 Arbeiten aus dem Bereich der Kalibrierung

Egorova, Simon, Wiesel, Gloye und Rojas [2005]: Haben eine automatische Kamerakalibrierung für das globale Kamerasystem in der Small Size League entwickelt. Dafür werden automatisch Farbabbildungen definiert und die Verzerrung des Bildes über detektierte Feldlinien berechnet. Für die Farbkalibrierung wird ein vordefiniertes Gitter verwendet, um Parameter für die Vorder- und Hintergrundtrennung zu finden. Nach dieser Vorverarbeitung lassen sich über die Farben die Konturen der Feldlinien detektieren. Die vollständige Kalibrierung erfolgt über eine Suche nach einer Transformation, deren Initialisierung über die Boundingbox des Feldes stattfindet. Die Konturpunkte werden über *biquadratische Interpolation* den Punkten im Weltmodell zugeordnet und die Parameter über Gradientenabstieg und eine Fehlerfunktion optimiert.
Die Kalibrierung über ein gegebenes Modell ist ein geeigneter Ansatz, aber durch unzureichende Formerkennung im System wurde in dieser Arbeit auf eine manuelle Kalibrierung gesetzt. Jedoch steckt auch hinter einem manuellen Verfahren ein gewisses Modell, welches in Kapitel 7 beschrieben wird.

Gunnarson, Wiesel und Raúl Rojas [2006]: Entwickelten ein automatisches Verfahren für die Farbkalibrierung, welches ein geometrisches Modell der Feldlinien in Weltkoordinaten und einen Ball in Bildkoordinaten erfordert. Das Verfahren trifft nur die Annahmen, dass das Feld, der Ball und die Tore grob unterscheidbare Farben haben und die Feldlinien heller als das restliche Feld sind. Für eine erfolgreiche Farbkalibrierung muss der Roboter ohne Farbannahmen lokalisiert werden, was über Kantendetektion geschieht. Entlang dieser Kanten werden die Feldlinien extrahiert und in ein Modell überführt, in dem dann Feldkomponenten identifiziert werden können. So lässt sich eine Lokalisierung bewerkstelligen, die nur durch die Symmetrieeigenschaft verfälscht sein kann. In durch Kanten gefundene Regionen lassen sich nun Startpunkte generieren, um Farbwachstum anzuwenden. Die Ballfarbe ist ein Sonderfall, bei dem die Farbschwellwerte vorsichtiger gesetzt werden und bei dem überprüft wird, ob die resultierende Region eine Ballform erreicht. Die dadurch entstandenen Regionen werden dann mit dem Feldlinienmodell verifiziert und im Falle der Tore durch die Farbe identifiziert und anschließend einer Seite zugeordnet. Für das Verfahren des Farbwachstums werden adaptive Schwellwerte verwendet, welche initial zufällig gewählt werden und nach erfolgreichen Iterationen bestehen bleiben oder nach fehlgeschlagenen Iterationen verworfen werden. Wenn die resultierenden Regionen zu groß werden, müssen die Schwellwerte ebenfalls verringert werden, während bei einer zu kleinen Region die Schwellwert vergrößert werden, um die Zielgröße zu erreichen.

Gunnarson, Wiesel und Raúl Rojas [2006] kalibrieren Farben über ein bekanntes Modell, um eine Klassifizierung zu ermöglichen. In dieser Arbeit werden Farben ohne Restriktionen durch ein Modell kalibriert, da durch eine gegebene Farbanalyse eine präzisere Erfassung für extrinsische Parameter bewerkstelligt werden soll. Um ein Verfahren ähnlich von Gunnarson, Wiesel und Raúl Rojas [2006] zu realisieren, müsste erst eine extrinsische Kalibrierung gegeben sein, um die Farbanalyse zu ermöglichen.

3.4 Arbeiten aus dem Bereich der Modellierung

Kwok und Fox [2005]: Versuchten auf dem Sony AIBO ein System zum Tracking eines Balles zu entwickeln. Dabei wurde eine gründliche Betrachtung des Filterproblems im Kontext der Ballhandhabung im RoboCup gegeben. Die Autoren entwickelten zu diesem Zweck einen Rao-Blackwellisierten Partikelfilter, welcher in der Lage ist, für die Vorhersage der Bewegung mehrere Dynamikmodelle zu verwenden. Dieser Ansatz kombiniert die Komplexität von Partikelfiltern mit der Effizienz und Genauigkeit von Kalman-Filtern. Das dafür entwickelte System integriert ein Großteil des Wissens über die RoboCup Domäne als Nichtlinearität, da der Ball vom Schiedsrichter anders platziert und durch Hindernisse abgefälscht werden kann. Dadurch steigt die Komplexität des Tracking-Problems, welches in der Robotik, den Autoren zufolge, von den folgenden Faktoren abhängig ist:

- Präzision der Schätzung der Eigenbewegung des Roboters
- Die Vorhersagbarkeit der Objektbewegungen
- Die Präzision des verwendeten Sensors

Die Autoren konkretisieren weiter diese Faktoren im Kontext des RoboCups:

- Nichtlineare Bewegung des Beobachters
- Physische Interaktion zwischen Ziel und Umgebung
- Physische Interaktion zwischen Beobachter und Ziel
- Unpräzise Messung und begrenzte Rechenzeit

Diese Entwicklung zeigt, dass das domänenspezifische Wissen in sorgfältig modellierter Form dem Roboter hilft, den Ball schneller zu finden, da die Interaktion mit der Umgebung und andere Eventualitäten berücksichtigt werden.
Kwok und Fox [2005] haben das Trackingproblem im RoboCup gründlich analysiert. Die Untersuchung zeigt, dass eine Kombination von Partikelfilter und Kalman-Filter für die Modellierung wünschenswert ist, wobei im Kontext dieser Arbeit das Dynamikmodell ein Spezialfall

ist.

Jochmann, Kerner, Tasse und Urbann [2012]: Entwickelten ein *Unscented Kalman-Filter*, welches mehrere Hypothesen beinhaltet. Dadurch sollten die Stärken von Partikelfilter und Kalman-Filter kombiniert werden. Dazu werden die Eigenschaften der jeweiligen Filter diskutiert, ihre Vor- und Nachteile herausgearbeitet und erläutert, wie diese sich durch den neuen Ansatz gegenseitig ergänzen. Ziel der Arbeit war es, diesen Ansatz in der SPL für eine bessere Lokalisierung zu verwenden. Dafür wird die aktuelle Schätzung als *Gaussian Mixture Model* repräsentiert, wodurch sich eine Änderung der Standard-Filter-Algorithmen ergibt, wie zum Beispiel eine zusätzliche Berechnung der Gewichte durch das Sensorupdate. Ebenso ist es in dem Verfahren möglich, die Wahrscheinlichkeit von False-Positives als Ausreißer zu modellieren. Zum Einstreuen von neuen Hypothesen werden Messungen verwendet. Dieser Ansatz bot damit in der SPL mehrere Vorteile, wie zum Beispiel die Verbesserung der Schätzung durch die Modellierung von False-Positives, die Möglichkeit, durch diese neue Filter-Methode das Kidnapped-Robot Problem zu lösen und das Erreichen einer Ausdrucksstärke, die dem Partikelfilter entspricht und dabei eine deutliche Laufzeitverbesserung bedeutet.
Die Arbeit benutzt ein zusammengefasstes Kalman-Filtering für mehrere Hypothesen, was zahlreiche Probleme der Lokalisierung im Kontext des RoboCup ausgleicht. Eine Behandlung von mehreren Schätzungen existiert auch in dieser Arbeit, jedoch wurde zu Gunsten der bekannten Kalman-Filter-Algorithmen auf eine Modellierung als *Gaussian Mixture Model* verzichtet.

Kapitel 4

Grundlagen

Dieses Kapitel behandelt einige Grundlagen, die für das vollständige Verständnis der nachfolgenden Kapitel essenziell sind.

4.1 Farbräume

Farbräume dienen dazu, das Verhalten realer Farben in der digitalen Welt zu simulieren. Dabei gilt: Je mehr Speicher für ein Bild benutzt wird, desto mehr Farben können dargestellt werden [vgl. Zeppenfeld, 2004, S. 178]. In den nachfolgenden Abschnitten werden sowohl der RGB-Farbraum als auch der HSV-Farbraum erklärt, welche in dieser Arbeit hauptsächlich verwendet werden.

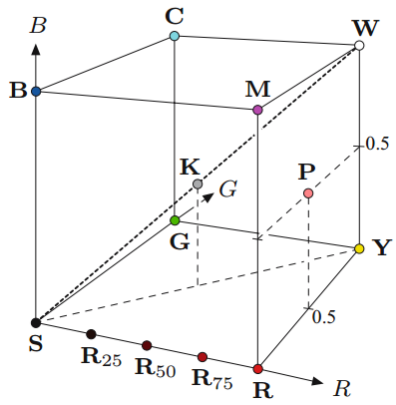
4.1.1 RGB

Das RGB-Modell ist ein physikalisch-technisches Farbmodell, welches die Primärfarben Rot, Grün und Blau mischt, um Farben zu erzeugen. Dabei lässt sich jede Farbe als Tripel (r, g, b) darstellen mit $r, g, b \in [0, 1]$. Dadurch lässt sich ein Koordinatensystem wie in Abb. 4.1 aufstellen und die genauen Farben wie in der daneben stehenden Tabelle ablesen. Dieses Modell findet Verwendung, da es der physikalischen Erstellung der Farben auf einem PC-Bildschirm entspricht [vgl. Zeppenfeld, 2004, S. 180].

4.1.2 HSV

Das HSV-Modell ist ein an der menschlichen Wahrnehmung orientiertes Modell, welches über den Farbton, die Sättigung und den nachfolgend erklärten Wert eine Farbe beschreibt [Zeppenfeld, 2004, S. 183].

- Farbton (**H**ue): Beschreibt, an welcher Stelle sich die Farbe im Farbspektrum befindet.



RGB-Werte

Pkt.	Farbe	*R*	*G*	*B*
S	Schwarz	0.00	0.00	0.00
R	Rot	1.00	0.00	0.00
Y	Gelb	1.00	1.00	0.00
G	Grün	0.00	1.00	0.00
C	Cyan	0.00	1.00	1.00
B	Blau	0.00	0.00	1.00
M	Magenta	1.00	0.00	1.00
W	Weiß	1.00	1.00	1.00
K	50% Grau	0.50	0.50	0.50
$\mathbf{R}_{75}$	75% Rot	0.75	0.00	0.00
$\mathbf{R}_{50}$	50% Rot	0.50	0.00	0.00
$\mathbf{R}_{25}$	25% Rot	0.25	0.00	0.00
P	Pink	1.00	0.50	0.50

Abbildung 4.1 Eine Visualisierung des RGB-Farbraumes mit einigen repräsentativen Werten, entnommen aus [Burger und Burge, 2006, S. 234].

- Sättigung (**S**aturation): Liegt zwischen 0 und 100, wobei 0 einer Graustufe entspricht und 100 einer leuchtenden Farbe.
- Wert (**V**alue): Synonym für die Helligkeit.

Auch dieses Modell lässt sich grafisch interpretieren, wie in Abb. 4.2 dargestellt. Zu sehen sind Rot (R), Grün (G), Blau (B), Gelb (Y), Zyan (C), Magenta (M), Schwarz (S) und Weiß (W). Auf Grund des Aufbaus des Farbraumes wird der Farbton meist in Grad spezifiziert, während Sättigung und Wert in Prozent angegeben werden.

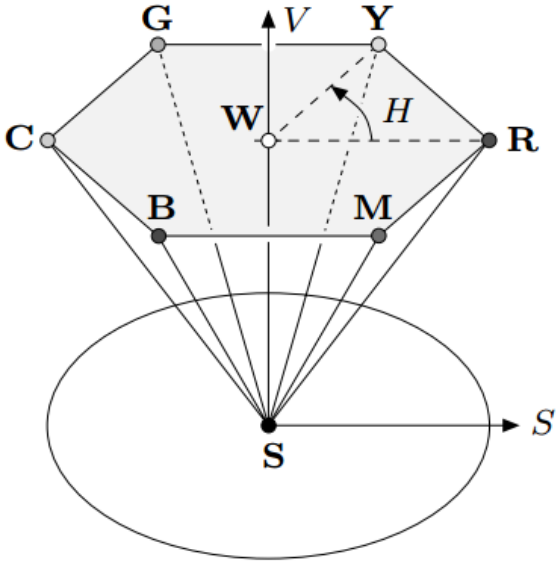


Abbildung 4.2 Visualisierung des HSV-Farbraumes, entnommen aus [Burger und Burge, 2006, S. 253].

4.2 Koordinatensysteme

Ein grundlegendes Konzept für diese Arbeit sind Koordinatensysteme. Ein Koordinatensystem mit n Dimensionen besteht aus n Achsen und einem Ursprung im Raum. Dabei lassen sich alle $n+1$ Komponenten als Vektoren darstellen, wobei die n Achsenvektoren orthogonal zueinander sein müssen. Dies führt dazu, dass man einen Punkt p in den Koordinaten eines Systems A in der Form von p^A beschreiben kann. Dadurch ist die mathematische Beziehung wie folgt im dreidimensionalen Fall als Transformation beschrieben [vgl. Frese, 2017]:

$$p = (p^A)_x \cdot A_x + (p^A)_y \cdot A_y + (p^A)_z \cdot A_z + (p^A)_w \cdot A_w \tag{4.1}$$
$$= A \cdot p^A \tag{4.2}$$

Dieses System gilt in beide Richtungen. Ein Beispiel hierfür findet sich im nächsten Unterabschnitt.

4.2.1 Das Weltkoordinatensystem

Das Weltkoordinatensystem im B-Human-Framework benutzt den Aufbau des Feldes, wie von RoboCup Technical Committee [2017, Kapitel 1] beschrieben. Die Achsenvektoren sind auf 1 mm normierte Vektoren entlang der beiden Seiten und der Ursprung des Systems ist die Mitte des Mittelkreises. Als dritte Koordinate, also Z-Achse, gibt es einen Vektor, der in die Luft zeigt und orthogonal auf den beiden anderen steht, jedoch wird diese Koordinate in den meisten Fällen zur Vereinfachung weggelassen, da wir uns in einer Ebene bewegen. Die anderen Achsen sind zur Veranschaulichung auf Abb. 4.3 vergrößert eingezeichnet. Diese Zeichnung der Koordinaten muss nicht unbedingt B-Human-konform sein, da das Koordinatensystem hier durch die Kalibrierung (vgl. Kapitel 7.3) gegeben ist und nicht durch die Spielrichtung des Teams.
Durch diesen Aufbau kann man nun für einen Punkt auf dem Feld die Koordinaten angeben. Will man zum Beispiel die Koordinaten der unteren rechten Ecke des Feldes in Abb. 4.3 durch die Formel 4.2 finden, so sieht die Gleichung folgendermaßen aus:

$$\begin{pmatrix} 4500\text{ mm} \\ 3000\text{ mm} \end{pmatrix} = \begin{pmatrix} A_x & A_y & A_w \end{pmatrix} \cdot p^A \tag{4.3}$$
$$= \begin{pmatrix} A_x & A_y & \vec{0} \end{pmatrix} \cdot p^A \tag{4.4}$$
$$= \begin{pmatrix} 1\text{ mm} & 0\text{ mm} & 0\text{ mm} \\ 0\text{ mm} & 1\text{ mm} & 0\text{ mm} \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{4.5}$$
$$\rightarrow \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 4500 \\ 3000 \end{pmatrix} \tag{4.6}$$

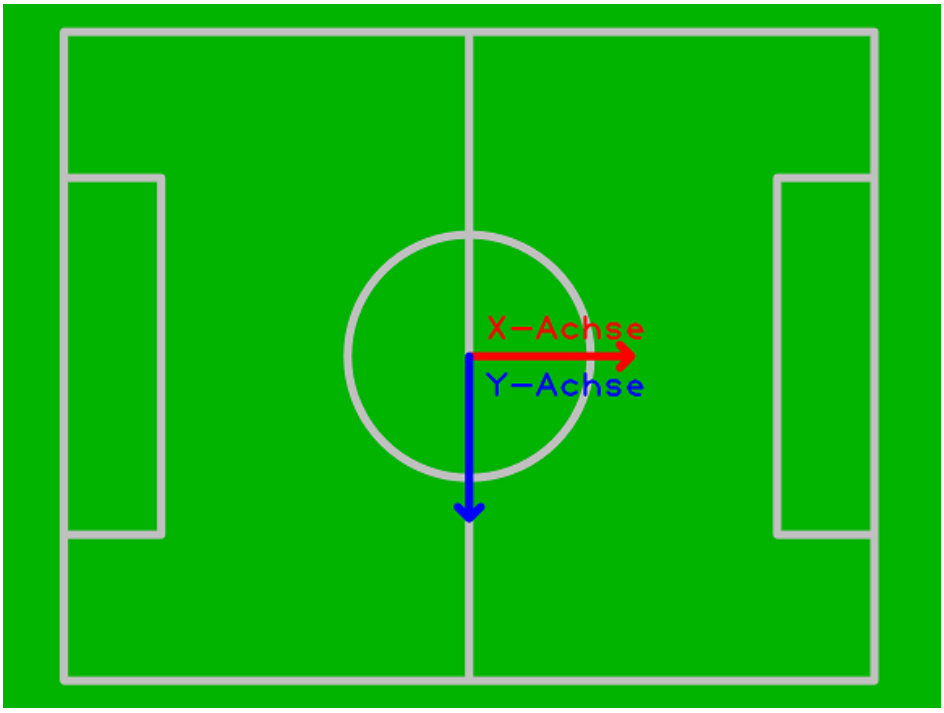


Abbildung 4.3 Darstellung der X- und Y-Achse im Weltkoordinatensystem.

Die generierten Daten dieser Arbeit werden im dargestellten Koordinatensystem gezeigt. Der Vorteil dieses Systems ist, dass es in der Realität leicht nachzuvollziehen ist.

4.2.2 Das Kamerakoordinatensystem

Dieses Koordinatensystem ist der logische Zwischenschritt zur Transformation der Weltkoordinaten zu Bildkoordinaten und vice versa. Der Aufbau des System ist in Abb. 4.4 illustriert. Die Besonderheit dieses Koordinatensystems besteht darin, dass der Ursprung in der Linse der Kamera sitzt. Die X- und Y-Achse sind im Sinne des Bildkoordinatensystems angeordnet, während die Z-Achse die Tiefe darstellt.

4.2.3 Das Bildkoordinatensystem

Dieses Koordinatensystem wird zur Beschreibung der Pixelpositionen in Bildern benutzt. Sämtliche Messungen im Bild sind initial in diesem System. Zur Veranschaulichung ist das System in Abb. 4.5 dargestellt. Die X-Koordinate gibt hier die Spalte im Bild an und die Y-Koordinate die Zeile. Dabei ist zu beachten, dass die Y-Achse nach unten zeigt, welches der seitenverkehrten Abbildung durch die Lochkamera geschuldet ist.

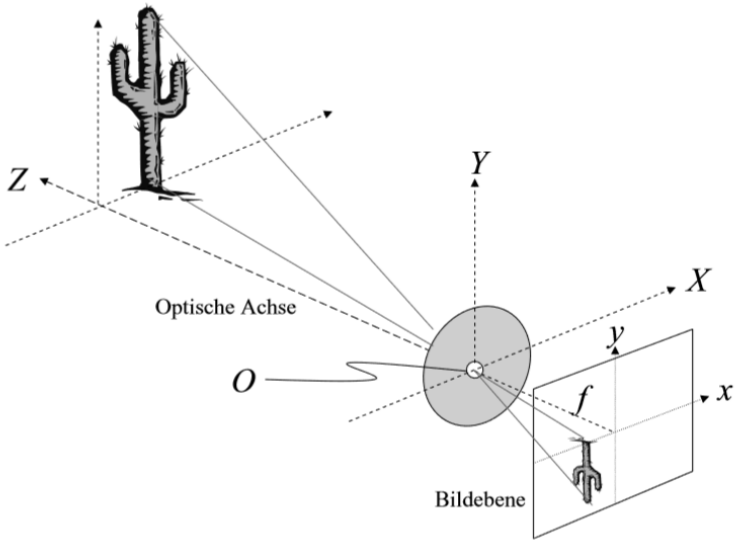


Abbildung 4.4 Visualisierung des Kamerakoordinatensystems im Kontext der Lochkamera, entnommen aus [Burger und Burge, 2006, S. 7].

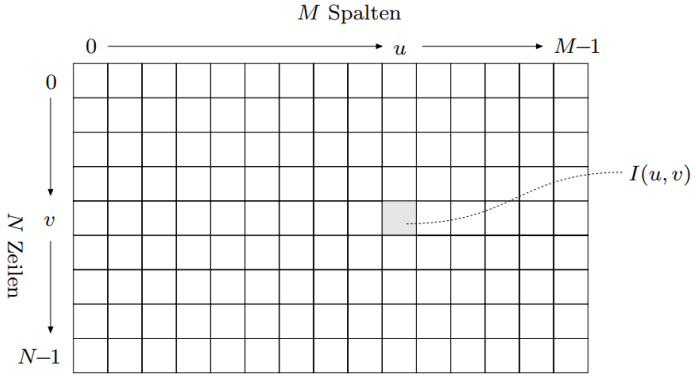


Abbildung 4.5 Darstellung des Bildkoordinatensystems, entnommen aus [Burger und Burge, 2006, S. 12].

4.3 Die Lochkamera

Das grundlegende Kameramodell in dieser Arbeit ist das Prinzip der Lochkamera. Eine Lochkamera ist ein geschlossener Raum mit einer sehr kleinen Öffnung an der Vorderseite und besitzt an der gegenüberliegenden Rückseite eine Bildebene. Durch diese Konstruktion werden Lichtstrahlen, die von einem Objekt ausgehend durch die Öffnung einfallen, geradlinig auf die Bildebene projiziert, wodurch ein verkleinertes und seitenverkehrtes Abbild der sichtbaren Szene entsteht. Dieses Prinzip ist in Abb. 4.4 dargestellt.
Wenn man annimmt, dass ein sichtbarer Objektpunkt in einer Distanz Z von der Lochebene und im vertikalen Abstand Y über der optischen Achse befindet, kann man folgende Berechnungen aufstellen:
Die Höhe der zugehörigen Projektion y wird durch zwei Parameter bestimmt und zwar die feste Tiefe der Kamerabox f, auch genannt Brennweite, und den Abstand Z des Objekts

vom Koordinatenursprung. Durch den Vergleich der ähnlichen Dreiecke ergibt sich der Zusammenhang proportional zur Tiefe der Box, der analog für die Breite der Projektion x und dem horizontalen Abstand X gilt:

$$y = -f\frac{Y}{Z} \tag{4.7}$$

$$x = -f\frac{X}{Z} \tag{4.8}$$

Die heutigen Kameras haben mit der Lochkamera und ihrer Lochblende bis auf das grundlegende optische Prinzip wenig gemeinsam. Damit man aber von Linsen sprechen kann, wird das Modell der „dünnen Linse“ als Verbesserung genommen, bei der die Lochblende durch eine Linse ersetzt wird, die als symmetrisch und unendlich dünn angenommen wird. Hier werden Lichstrahlen an einer virtuellen Ebene in der Linsenmitte gebrochen. Die Abbildungsgeometrie bleibt in diesem Modell aber unverändert, wie in Abb. 4.6 dargestellt. Auch dieses Modell bietet keine vollständige Beschreibung von Linsensystemen, da Schärfe, Blenden, geometrische Verzerrungen und viele andere Effekte nicht modelliert werden [vgl. Burger und Burge, 2006, S. 5-9].

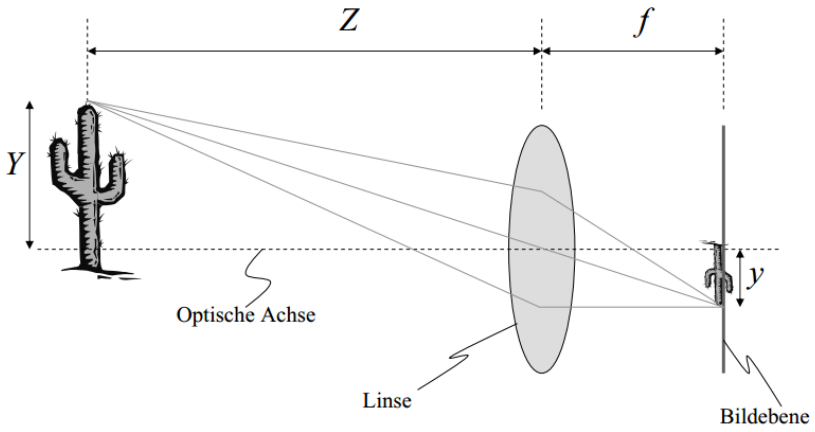


Abbildung 4.6 Darstellung des Linsenmodells, entnommen aus [Burger und Burge, 2006, S. 8].

Wichtig an diesem Modell ist, dass dadurch Transformationen bzw. Projektionen von der realen Welt auf die Bildebene beschrieben werden. Diese Transformationen werden in Kapitel 7.4 genauer erklärt.

4.4 Die Normalverteilung

Eine Normalverteilung oder auch Gauß-Verteilung beschreibt eine Zufallsvariable X über die folgende Dichtefunktion:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \cdot e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}, \quad \sigma > 0 \tag{4.9}$$

Diese Verteilung $X \sim \mathcal{N}(\mu, \sigma^2)$ beschreibt als Graph eine Glocke, siehe Abb. 4.7, weshalb sie auch als Gauß'sche Glockenkurve bezeichnet wird, und wird nur durch ihre Parameter geregelt:

- μ: Der Erwartungswert, der die Mitte der Verteilung darstellt.
- σ: Die Standardabweichung, welche die Breite der Glocke beschreibt.

Dabei erfüllt die Funktion der Normalverteilung zwei Eigenschaften [G. Teschl und S. Teschl, 2007, S. 309 - 310]:

- Die Funktion hat einen symmetrischen Aufbau um die Mitte μ und zwei Wendepunkte an den Stellen $\mu \pm \sigma$.
- Der Flächeninhalt unter der Kurve ist gleich 1.

Durch ihre Eigenschaften wird diese Verteilung auch in der Robotik interessant, da man Schätzungen mit einer Wahrscheinlichkeit $f(x)$, dem wahrscheinlichsten Zustand μ und einer Unsicherheit σ beschreiben kann. Zum Beispiel kann man damit die Schätzung einer Roboterposition angeben, wobei μ die Position als Vektor beschreiben würde und σ die Unsicherheit entlang jeder Achse. Wie dieses Beispiel andeutet können Normalverteilungen auch mehrdimensional sein.

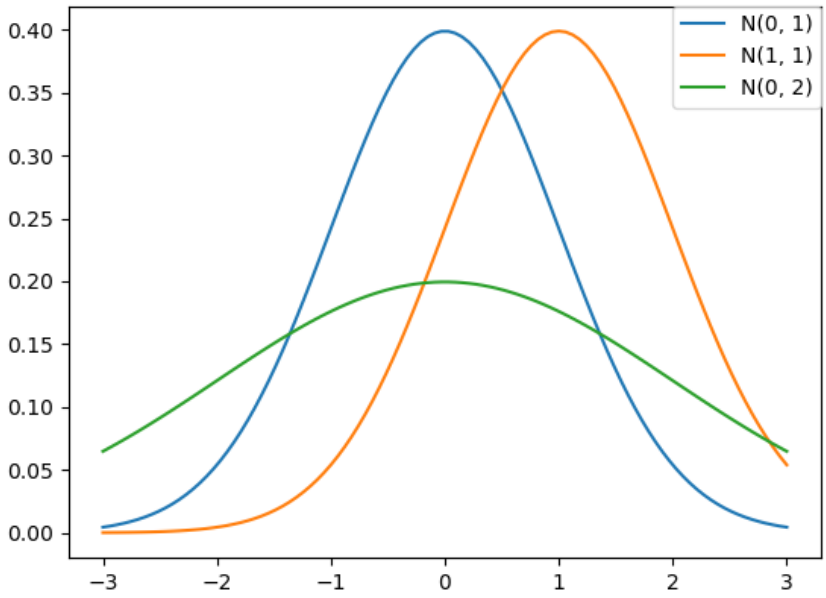


Abbildung 4.7 Darstellung verschiedener Normalverteilungen.

4.4.1 Das *Gaussian Mixture Model*

Ein *Gaussian Mixture Model* (GMM) ist eine parametrisierte Wahrscheinlichkeitsfunktion, die sich aus einer gewichteten Summe von Gauß-Verteilungen zusammensetzt. Dabei kann

man sie mit ihren n Gauß-Komponenten durch die folgende Funktion beschreiben:

$$p(x \mid \lambda) = \sum_{i=1}^{n} w_i \, f(x \mid \mu_i, \Sigma_i) \tag{4.10}$$

Dabei gilt:

$$\lambda = \{w_i, \mu_i, \Sigma_i\}, \qquad i = 1, \ldots, n \tag{4.11}$$

$$\sum_{i=1}^{n} w_i = 1 \tag{4.12}$$

Durch diese Modellierung lassen sich auch wesentlich komplexere Verteilungen beschreiben, wie in Abb. 4.8 dargestellt [vgl. Reynolds, 2015].

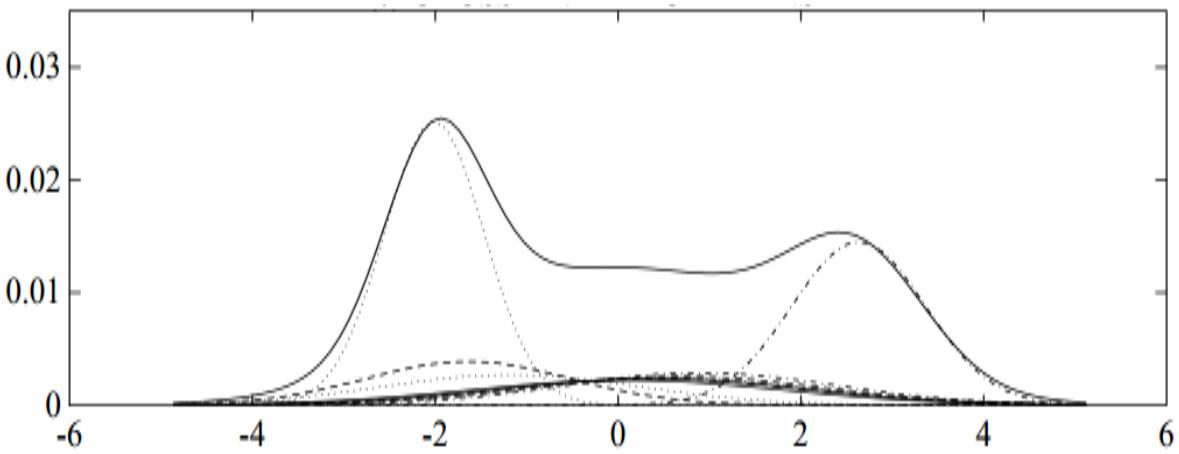


Abbildung 4.8 Eine beispielhafte Verteilung dargestellt als *Gaussian Mixture Model*, entnommen aus [Reynolds, 2015, S. 2].

Für diese Arbeit sind zwei weitere Eigenschaften eines *Gaussian Mixture Model* interessant.

1. Die Berechnung der wahrscheinlichsten Komponente wie in Formel 4.13 kann zur Integration von Messungen genutzt werden.

2. Das parametrisierbare Modell kann durch iterative Erwartungsmaximierung als unüberwachtes Lernverfahren genutzt werden. Ein Beispiel, was ein solches Verfahren bewirken kann, ist in Abb. 4.9 illustriert, bei dem die Verteilung an vorhandene Trainingsdaten angepasst wird.

$$\mathcal{N}_i = \arg\max_i \frac{w_i \, f(x \mid \mu_i, \Sigma_i)}{\sum_{k=1}^{n} w_k \, f(x \mid \mu_k, \Sigma_k)} \tag{4.13}$$

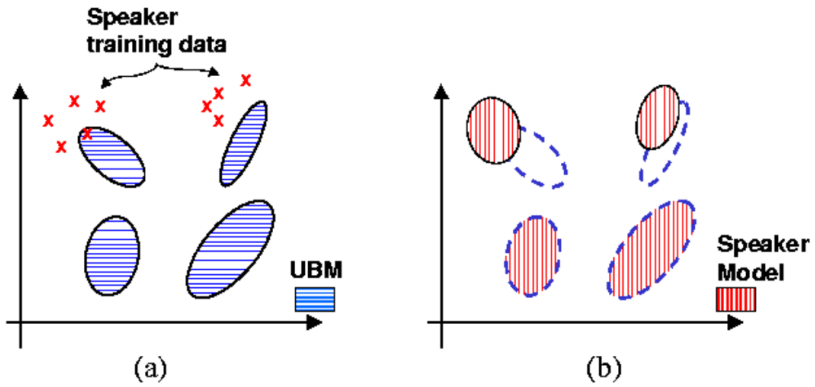


Abbildung 4.9 Beispiel einer Erwartungsmaximierung auf einem *Gaussian Mixture Model*. In (a) werden die Trainingsdaten mit der vorhandenen Verteilung abgeglichen und in (b) die Verteilung in die entsprechende Richtung korrigiert. Entnommen aus [Reynolds, 2015, S. 5].

4.5 Das Kalman-Filter

Das Kalman-Filter ist ein Verfahren zum Filtern und Vorhersagen linearer Systeme, wobei es ein Schätzmodell in einem kontinuierlichen Zustandsraum aufstellt. Dabei wird eine Schätzung x_t als eine Gauß-Verteilung zu einem Zeitpunkt t dargestellt. Das Verfahren besitzt zu Anwendungszwecken vier Annahmen, die erfüllt sein müssen:

1. Die Markov Annahme ist erfüllt. Das bedeutet, dass der jetzige Zustand nur von seinem direkten Vorgängerzustand abhängig ist.
2. Die Zustandsübergangswahrscheinlichkeitsfunktion $p(x_t \mid u_t, x_{t-1})$ ist linear in ihren Argumenten mit einem gauß'schen Fehler $R_t \sim \mathcal{N}(0, \sigma_{r_t}^2)$.
3. Die Messwahrscheinlichkeitsfunktion ist linear in ihren Argumenten mit einem gauß'schen Fehler $Q_t \sim \mathcal{N}(0, \sigma_{q_t}^2)$.
4. Die anfängliche Schätzung x_0 ist eine Normalverteilung $x_0 \sim \mathcal{N}(\mu_0, \Sigma_0)$.

Diese Annahmen sorgen dafür, dass das gesamte Verfahren valide ist und die Schätzungen durchgängig Normalverteilungen sind. Die Anwendung des Kalman-Filters mit Dynamik- und Messschritt ist in Algorithmus 1 dargestellt [vgl. Thrun, Burgard und Fox, 2006, S. 40 - 42]. Dieser Algorithmus enthält viele Variablen, die es hier in eigenen Worten zu erklären gilt:

- $\mathcal{N}(\mu_{t-1}, \Sigma_{t-1})$ gibt die Ausgangsschätzung mit dem Mittelwert μ_{t-1} und der Unsicherheit in Form einer Kovarianzmatrix Σ_{t-1} an.
- u_t ist die Zustandsübergangsmessung.
- z_t ist die Zustandsmessung, die im Gegensatz zu u_t den Zustand als Ganzes beschreiben kann.

Algorithmus 1 Kalman-Filter($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$)

Dynamikschritt:

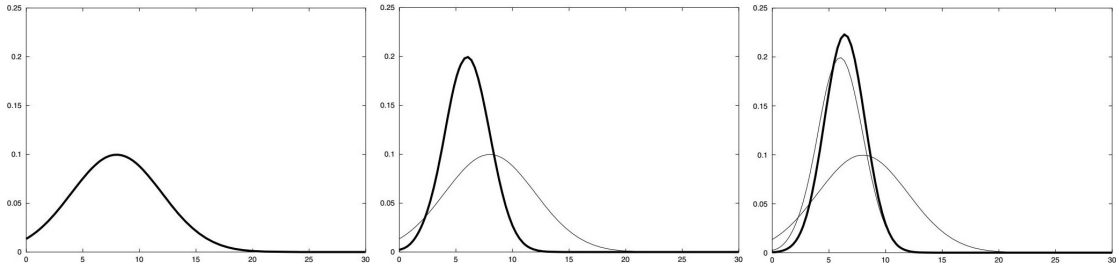
1: $\bar{\mu}_t = A_t\, \mu_{t-1} + B_t\, u_t$
2: $\bar{\Sigma}_t = A_t\, \Sigma_{t-1}\, A_t^T + R_t$

Messschritt:

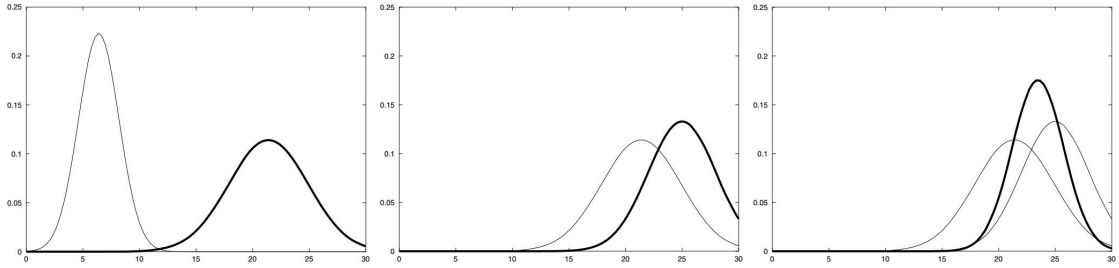
3: $K_t = \bar{\Sigma}_t\, C_t^T\, (C_t\, \bar{\Sigma}_t\, C_t^T + Q_t)^{-1}$
4: $\mu_t = \bar{\mu}_t + K_t\, (z_t - C_t\, \bar{\mu}_t)$
5: $\Sigma_t = (I - K_t\, C_t)\, \bar{\Sigma}_t$
6: **return** μ_t, Σ_t

- A_t beschreibt die lineare Abhängigkeit von dem Ausgangszustand.
- B_t beschreibt das Dynamikmodell, mit dem man aus μ_{t-1} und u_t den Folgezustand μ_t berechnet.
- R_t ist der gauß'sche Fehler, der durch die Zustandsübergangsmessung in der Form $R_t \sim \mathcal{N}(0, \sigma_{r_t}^2)$ gegeben ist.
- $\mathcal{N}(\bar{\mu}_t, \bar{\Sigma}_t)$ beschreibt die Schätzung nach Anwendung des Dynamikmodells.
- K_t ist der Kalman Gain, der die Integration der aktuellen Messung in den Zustand gewichtet.
- C_t ist eine Matrix, die den aktuellen Zustand in den Messraum transformiert.
- Q_t ist der gauß'sche Fehler, der durch die Zustandsmessung in der Form $Q_t \sim \mathcal{N}(0, \sigma_{q_t}^2)$ gegeben ist.
- I ist eine Identitätsmatrix, die zur Vereinfachung der Formel ergänzt wurde.
- $\mathcal{N}(\mu_t, \Sigma_t)$ beschreibt die aktuelle Schätzung nach Anwendung des Messmodells.

In Abb. 4.10 wird die Funktionsweise eines Kalman-Filters visualisiert.



(a) Die initiale Schätzung als Normalverteilung (b) Eine Messung in Form einer Normalverteilung (c) Integration der Messung in die bestehende Schätzung



(d) Die Schätzung nach einer Bewegung (e) Eine neue Messung (f) Die resultierende Schätzung

Abbildung 4.10 Illustration einer Filterung über ein Kalman-Filter bei dem die Schätzung und die Messungen als Normalverteilungen vorliegen. Die Schätzung erfährt eine Verringerung der Unsicherheit durch Messungen und eine Vergrößerung durch Bewegungen. Entnommen aus [Thrun, Burgard und Fox, 2006, S. 44].

4.5.1 Das *Unscented Kalman-Filter*

Das Kalman-Filter ist generell für lineare Systeme geeignet, aber es gibt auch Erweiterungen des Modells, die nichtlineare Systeme filtern. Eine bekannte Variante ist der Extended Kalman Filter, welcher eine Linearisierung über Taylor-Reihen-Erweiterung der Verteilungen durchführt. Da es für manche Anwendungen schwer ist, Ableitungen von komplexen Modellen zu erstellen, wurde das *Unscented Kalman-Filter* entwickelt.

Das *Unscented Kalman-Filter* (UKF) linearisiert über sogenannte Sigmapunkte, welche Repräsentanten der aktuellen Verteilung sind, und übergibt diese der Zustandsübergangsfunktion g oder der Messfunktion h. Um letztendlich eine Schätzung zu erhalten, werden diese Punkte gemittelt.

Die $2n+1$ Sigmapunkte, wobei n die Anzahl der Dimensionen ist, werden nach dem folgenden Muster gewählt:

$$\mathcal{X}^{[0]} = \mu \tag{4.14}$$

$$\mathcal{X}^{[i]} = \mu + (\sqrt{(n+\lambda)\,\Sigma})_i, \text{ für } i = 1, \ldots, n \tag{4.15}$$

$$\mathcal{X}^{[i]} = \mu - (\sqrt{(n+\lambda)\,\Sigma})_{i-n}, \text{ für } i = n+1, \ldots, 2n \tag{4.16}$$

$$\text{mit } \lambda = \alpha^2(n+\kappa) - n \tag{4.17}$$

Hierbei sind α und κ Parameter, um den Abstand zum Mittelpunkt zu regulieren. Zu jedem Sigmapunkt werden zwei Gewichtungsfaktoren mit geführt, welche angeben, wie stark der Einfluss bei der Summierung für einen neuen Mittelwert oder Kovarianz ist:

$$w_m^{[0]} = \frac{\lambda}{n+\lambda} \tag{4.18}$$

$$w_c^{[0]} = \frac{\lambda}{n+\lambda} + (1 - \alpha^2 + \beta) \tag{4.19}$$

$$w_m^{[i]} = w_c^{[i]} = \frac{1}{2(n+\lambda)}, \text{ für } i = 1, \ldots, 2n \tag{4.20}$$

Dabei kann der Parameter β genutzt werden, um zusätzliches Wissen über die unterliegende Normalverteilung einzuspeisen. Wenn die Verteilung aber eine reine Normalverteilung ist, gilt $\beta = 2$. Der Algorithmus des UKF ist in Algorithmus 2 dargestellt [vgl. Thrun, Burgard und Fox, 2006, S. 65 - 70].

Algorithmus 2 Unscented Kalman-Filter($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$)

Dynamikschritt:

1: $\mathcal{X}_{t-1} = (\mu_{t-1} \quad \mu_{t-1} + \gamma\sqrt{\Sigma_{t-1}} \quad \mu_{t-1} - \gamma\sqrt{\Sigma_{t-1}})$

2: $\bar{\mathcal{X}}_t^* = g(u_t, \mathcal{X}_{t-1})$

3: $\bar{\mu}_t = \sum_{i=0}^{2n} w_m^{[i]} \, \bar{\mathcal{X}}_t^{*[i]}$

4: $\bar{\Sigma}_t = \sum_{i=0}^{2n} w_c^{[i]} (\bar{\mathcal{X}}_t^{*[i]} - \bar{\mu}_t)(\bar{\mathcal{X}}_t^{*[i]} - \bar{\mu}_t)^T + R_t$

Messschritt:

5: $\bar{\mathcal{X}}_t = (\bar{\mu}_t \quad \bar{\mu}_t + \gamma\sqrt{\bar{\Sigma}_t} \quad \bar{\mu}_t - \gamma\sqrt{\bar{\Sigma}_t})$

6: $\bar{\mathcal{Z}}_t = h(\bar{\mathcal{X}}_t)$

7: $\hat{z}_t = \sum_{i=0}^{2n} w_m^{[i]} \bar{\mathcal{Z}}_t^{[i]}$

8: $S_t = \sum_{i=0}^{2n} w_c^{[i]} (\bar{\mathcal{Z}}_t^{[i]} - \hat{z}_t)(\bar{\mathcal{Z}}_t^{[i]} - \hat{z}_t)^T + Q_t$

9: $\bar{\Sigma}_t^{x,z} = \sum_{i=0}^{2n} w_c^{[i]} (\bar{\mathcal{X}}_t^{[i]} - \bar{\mu}_t)(\bar{\mathcal{Z}}_t^{[i]} - \hat{z}_t)^T$

10: $K_t = \bar{\Sigma}_t^{x,z} \, S_t^{-1}$

11: $\mu_t = \bar{\mu}_t + K_t(z_t - \hat{z}_t)$

12: $\Sigma_t = \bar{\Sigma}_t - K_t \, S_t \, K_t^T$

13: **return** μ_t, Σ_t

Mit diesen Beschreibungen werden einige Erklärungen zu den neuen Variablen benötigt, die sich stark von der linearen Kalman-Filter-Form unterscheiden:

- γ ist eine Abkürzung der Parametrisierung und bedeutet $\gamma = \sqrt{n + \lambda}$.
- g ist die Zustandsübergangsfunktion, welche das Dynamikmodell beinhaltet, und muss nicht linear sein.
- h ist die Messfunktion, welche das Messmodell beinhaltet, und muss nicht linear sein.
- S_t beinhaltet die Unsicherheit, welche durch die Anwendung von h entsteht.
- $\Sigma_t^{x,z}$ beschreibt, wie stark Zustand und Beobachtung gemeinsam schwanken.

In dieser Arbeit wird allerdings ein UKF verwendet bei dem alle Gewichte gleich groß sind, bei dem die Aufsummierung eine Mittelung bewirkt und die Entfernung der Sigmapunkte vom Mittelpunkt genau der Choleskyzerlegung entspricht. In Formeln bedeutet dies:

$$\gamma = 1 \tag{4.21}$$

$$w_m^{[i]} = \frac{1}{2n+1}, \qquad \forall i \in [0, \ldots, 2n] \tag{4.22}$$

$$w_c^{[i]} = \frac{1}{2}, \qquad \forall i \in [0, \ldots, 2n] \tag{4.23}$$

4.6 Kantendetektion

In dieser Abschlussarbeit werden Kantenbilder zur Detektion von Robotern benutzt. Um aber die Berechnung dieser Bilder vollständig zu durchdringen, muss man sich mit einigen Grundbegriffen vertraut machen.

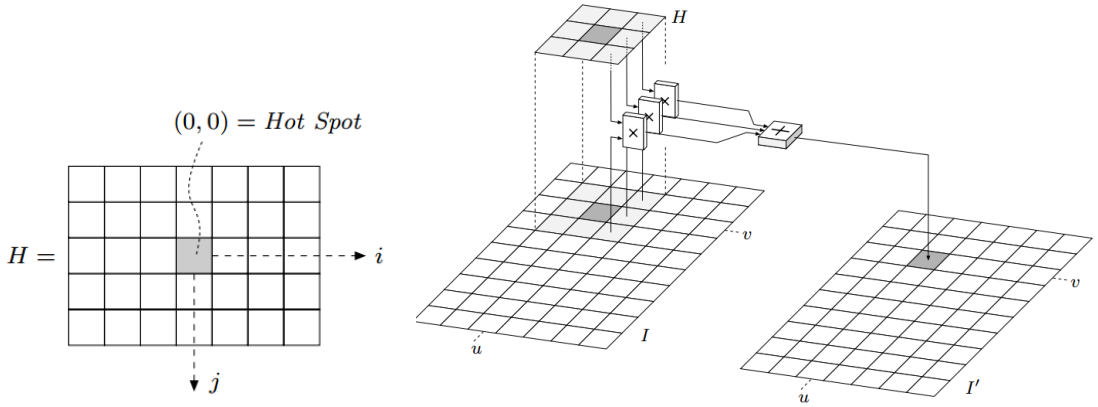
4.6.1 Lineare Filter

Dieser Mechanismus wird so genannt, da er Pixelwerte innerhalb einer Region durch eine gewichtete Summe verknüpft. Dadurch können komplexere Funktionen auf Bilder beschrieben werden, die mehrere Pixel als Eingabe benutzen. Ein gutes Beispiel hierfür ist der 3×3 Glättungsfilter, der einen Durchschnitt über alle neun Pixel im Ursprungsbild bildet und den entsprechenden Pixel im neuen Bild auf den erhaltenen Wert setzt. Die Filtermatrix sieht hierfür folgendermaßen aus:

$$H(i,j) = \begin{bmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{bmatrix} = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{4.24}$$

Eine solche Filtermatrix besitzt ein eigenes Koordinatensystem und wird eingesetzt, indem man diese auf alle Pixel des Ursprungsbildes anwendet, sodass alle Pixel im Filter einen Wert

haben. Hierdurch werden die äußeren Pixel im Bild nicht durch den Filter selbst abgebildet. Das Koordinatensystem und die Anwendung eines Filters sind in Abb. 4.11 illustriert.



(a) Das Koordinatensystem (b) Die Anwendung

Abbildung 4.11 Das Bild (a) zeigt das Koordinatensystem, mit dem man sich im Filter orientiert und (b) zeigt das Prinzip der Anwendung. Entnommen aus [Burger und Burge, 2006, S. 92].

Ein Filter ist somit, wie das Bild selbst, eine Abbildung $H : \mathbb{Z} \times \mathbb{Z} \to \mathbb{R}$. Die Anwendung des Filters, in dem der Ursprung des Koordinatensystems H auf dem aktuellen Bildpunkt $I(u, v)$ ist, ist mathematisch definiert als:

$$I'(u,v) = \sum_{(i,j) \in R} I(u+i, v+j) \cdot H(i,j) \tag{4.25}$$

Hierbei stellt R die Menge aller möglichen Koordinaten innerhalb von H dar [vgl. Burger und Burge, 2006, S. 91 - 93].
Lineare Filter sind als Operation in der Mathematik unter der Bezeichnung „lineare Faltungen“ bekannt, bei der zwei Funktionen gleicher Dimensionalität verknüpft werden [vgl. Burger und Burge, 2006, S. 101].
Dieses Konstrukt ermöglicht in der Bildverarbeitung eine Menge interessanter Verfahren und insbesondere für diese Arbeit die Kantendetektion.

4.6.2 Gradiententheorie auf Bildern

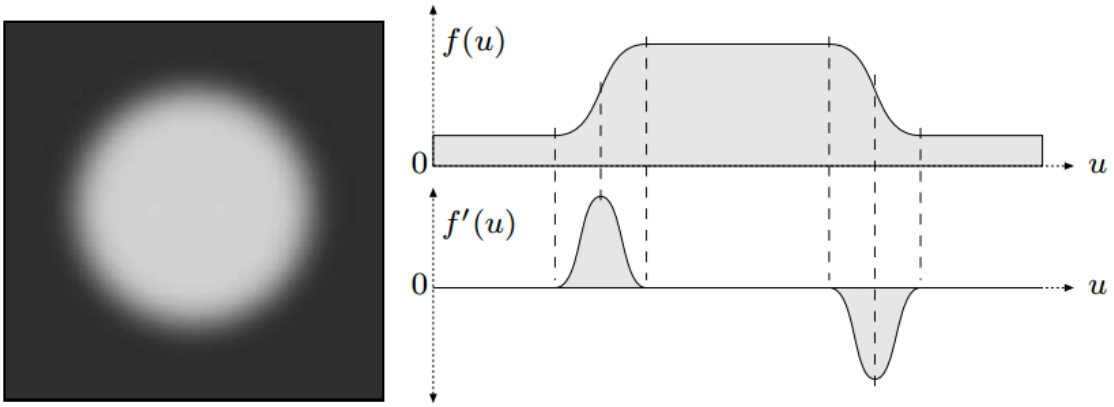
Für die Erkennung von Objekten spielen nicht nur Farben eine Rolle, sondern auch Formen. Um diese Formen zu konstruieren, muss man auch Kanten wahrnehmen. Eine Kante kann man als eine Änderung der Intensität entlang einer Richtung des Bildes auf einem kleinen Raum definieren, was man auch in Form einer Ableitung beschreiben kann. Als Vereinfachung kann man die Ableitung

$$f'(u) = \frac{df}{du}(u) \tag{4.26}$$

entlang einer Dimension eines Grauwertbildes betrachten. In Abb. 4.12 ist in (a) ein Beispielbild gegeben, dessen mittlere Zeile als Bildfunktion in (b) zusammen mit ihrer Ableitung dargestellt ist. Da die Bildfunktion nur als Intensitätsverlauf existiert, kann man die Ableitung über Tangenten der Nachbarpunkte approximieren, wie in Formel 4.27.

$$\frac{df}{du}(u) \approx \frac{f(u+1) - f(u-1)}{2} = 0,5 \cdot (f(u+1) - f(u-1)) \tag{4.27}$$

Wie in Abb. 4.12 (b) zu sehen, erzeugen Kanten einen starken Ausschlag in eine beliebige Richtung. Diese Eigenschaft macht man sich für die Kantenerkennung zunutze, doch man darf nicht vergessen, dass man sich im zweidimensionalen Raum befindet, bei dem man mit Ableitungen mehrere Komponenten betrachten muss.



(a) Das Graustufenbild (b) Die Bildfunktion und ihre Ableitung

Abbildung 4.12 Veranschaulichung einer Bildableitung, entnommen aus [Burger und Burge, 2006, S. 118].

Da ein Bild aus mehr als einer Dimension besteht, arbeitet man mit den zwei partiellen Ableitungen,

$$\frac{\partial I}{\partial u}(u,v) \quad \text{und} \quad \frac{\partial I}{\partial v}(u,v) \tag{4.28}$$

die die Ableitung an einer Stelle (u, v) im Bild beschreiben. Daher werden sie auch als Gradientenvektor zusammengefasst

$$\nabla I(u,v) = \begin{bmatrix} \frac{\partial I}{\partial u}(u,v) \\ \frac{\partial I}{\partial v}(u,v) \end{bmatrix} \tag{4.29}$$

und der Betrag des Gradienten wird wie folgt berechnet:

$$|\nabla I| = \sqrt{\left(\frac{\partial I}{\partial u}\right)^2 + \left(\frac{\partial I}{\partial v}\right)^2} \tag{4.30}$$

Um nun eine gut anwendbare Kantenberechnung zu ermöglichen, wird auf das Prinzip der Tangente von Formel 4.27 in Kombination mit linearen Filtern zurückgegriffen. Dadurch lassen sich die Ableitungen der beiden Richtungen als Koeffizientenmatrizen darstellen [vgl. Burger und Burge, 2006, S. 117 - 120]:

$$H_x^D = \begin{bmatrix} -0.5 & 0 & 0.5 \end{bmatrix} = 0.5 \cdot \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} \tag{4.31}$$

$$H_y^D = \begin{bmatrix} -0.5 \\ 0 \\ 0.5 \end{bmatrix} = 0.5 \cdot \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} \tag{4.32}$$

4.6.3 Der Sobel-Operator

Der Sobel-Operator ist eine Adaption des Prewitt-Operators, der über drei Zeilen und Spalten arbeitet, um nicht so anfällig für Rauschen zu sein wie der einfache Gradientenoperator. Dabei wird über neun Pixel gemittelt, wobei beim Sobel die Mitte eine stärkere Gewichtung erhält, wie in Formel 4.33 in zu sehen ist.

$$\begin{aligned} H_x^S &= \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \\ H_y^S &= \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \end{aligned} \tag{4.33}$$

Die Anwendung dieser Operatoren entspricht nach einer Skalierung einer Schätzung des lokalen Bildgradienten:

$$\nabla I(u,v) \approx \frac{1}{8} \begin{bmatrix} H_x^S * I \\ H_y^S * I \end{bmatrix} \tag{4.34}$$

Für einen minimierten Winkelfehler wird eine verbesserte Form des Sobel-Operators vorgeschlagen:

$$\begin{aligned} H_x^{S'} &= \frac{1}{32} \begin{bmatrix} -3 & 0 & 3 \\ -10 & 0 & 10 \\ -3 & 0 & 3 \end{bmatrix} \\ H_y^{S'} &= \frac{1}{32} \begin{bmatrix} -3 & -10 & -3 \\ 0 & 0 & 0 \\ 3 & 10 & 3 \end{bmatrix} \end{aligned} \tag{4.35}$$

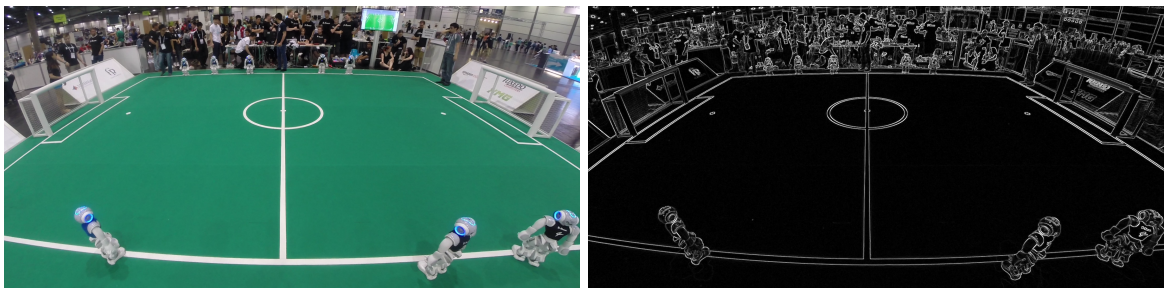
Unabhängig vom verwendeten Filters kann man die folgenden Resultate generieren:

$$\text{Die Filtergebnisse:} \quad D_x(u,v) = H_x * I \text{ und } D_y(u,v) = H_y * I \tag{4.36}$$

$$\text{Die Kantenstärke:} \quad E(u,v) = \sqrt{(D_x(u,v))^2 + (D_y(u,v))^2} \tag{4.37}$$

$$\text{Die lokale Kantenrichtung:} \quad \Phi(u,v) = \tan^{-1}\left(\frac{D_y(u,v)}{D_x(u,v)}\right) \tag{4.38}$$

Der Sobel-Operator ist aufgrund guter Ergebnisse und Einfachheit sehr verbreitet [vgl. Burger und Burge, 2006, S. 121 - 123].
In dieser Arbeit wird eine Kombination der beiden Sobel-Operatoren aus der Bibliothek *OpenCV* verwendet, mit der sich gute Ergebnisse erzielen lassen, wie in Abb. 4.13 dargestellt.



(a) Das Originalbild (b) Das Kantenbild

Abbildung 4.13 Ein Beispielbild aus einem Spiel (a), aus dem in (b) ein Kantenbild erstellt wurde.

4.7 Der FAST Algorithmus

Der „Features from Accelerated Segment Test“ (FAST) Algorithmus untersucht einen Eckenkandidaten p, indem er einen Kreis aus 16 Pixeln um ihn herum betrachtet. Der Kandidat p wird mit seiner Intensität I_p als Ecke klassifiziert, wenn es eine Pixelmenge P_c der Größe n von aneinander grenzenden Pixeln in dem Kreis gibt, sodass P_c eines der folgenden Kriterien erfüllt:

$$\forall x \in P_c : I_x > I_p + t \tag{4.39}$$

$$\forall x \in P_c : I_x < I_p - t \tag{4.40}$$

t ist ein Schwellwert, um zu regulieren, wie viele Features gefunden werden sollen. In der Umsetzung wird $n = 12$ gewählt, damit man in vier Kompassrichtungen eine schnelle Überprüfung machen kann, da, wenn die im Beispiel Abb. 4.14 ausgewählten Pixel 1, 5, 9 und 13 betrachtet werden, mindestens drei von ihnen eines der oben genannten Kriterien erfüllen müssen. Der vollständige Test kann dann auf die übrigen Kandidaten angewendet werden.

Dieser Eckenerkenner ist schnell, kann dafür aber mehrere Schwächen haben [vgl. Rosten und Drummond, 2006, S. 433]:

1. Der Schnelltest lässt sich nicht generell für $n < 12$ anwenden.
2. Die Wahl der Anordnung der Testpixel trifft eine indirekte Annahme des Feature-Vorkommens.
3. Das Wissen der ersten 4 Tests geht verloren.
4. Es werden mehrere benachbarte Features erkannt.

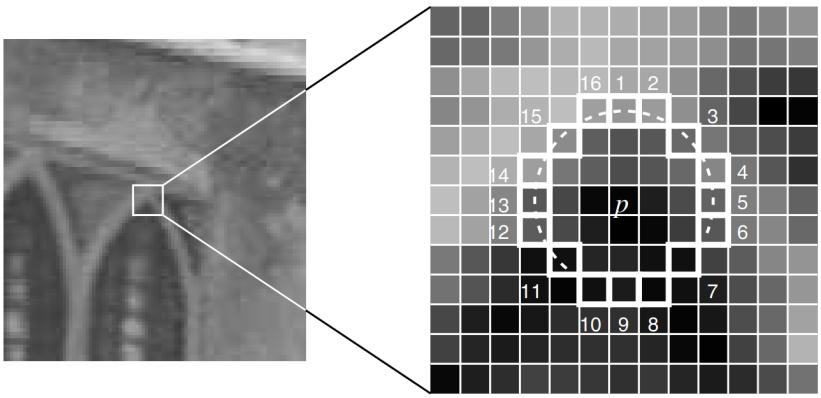


Abbildung 4.14 Beispiel einer FAST Anwendung, entnommen aus [Rosten und Drummond, 2006, S. 434].

4.8 Gruppentheorie zur Generierung von Zahlen

Eine Gruppe ist ein Konstrukt, um Zahlen in einem gewissen Wertebereich zu generieren. Dieses Prinzip bedarf einiger Definitionen und Erläuterungen, die hier aufgelistet werden. Die Motivation hinter diesem Konstrukt wird erst in Kapitel 6.3 erklärt.

4.8.1 Definition einer Gruppe

Eine Gruppe besteht aus einer Menge von Elementen G und einer Operation $\circ$, die man auf zwei Elemente anwenden kann. Eine Gruppe hat folgende Eigenschaften [vgl. Paar und Pelzl, 2010, S. 209]:

1. Die Gruppenoperation $\circ$ ist geschlossen. Dies bedeutet:

$$a \circ b = c \in G \qquad \forall a, b \in G \tag{4.41}$$

2. Die Gruppenoperation ist assoziativ:

$$a \circ (b \circ c) = (a \circ b) \circ c \qquad \forall a, b, c \in G \tag{4.42}$$

3. Es gibt ein Element $1 \in G$, welches als neutrales Element oder Identitätselement bezeichnet wird, bei dem gilt:

$$a \circ 1 = 1 \circ a = a \qquad \forall a \in G \tag{4.43}$$

4. Für jedes Element $a \in G$ existiert ein inverses Element $a^{-1} \in G$, so dass gilt:

$$a \circ a^{-1} = a^{-1} \circ a = 1 \tag{4.44}$$

5. Eine Gruppe ist abelsch oder auch kommutativ, wenn gilt:

$$a \circ b = b \circ a \qquad \forall a, b \in G \tag{4.45}$$

4.8.2 Erweiterte Gruppendefinitionen

Um gewisse Eigenschaften präzise beschreiben zu können, werden weitere Definitionen ergänzt [vgl. Paar und Pelzl, 2010, S. 211 - 213]:

- Eine Gruppe $(G, \circ)$ ist eine endliche Gruppe, wenn eine endliche Anzahl von Elementen in G enthalten ist, also $|G| < \infty$.
- Die Ordnung $ord(a)$ eines Elementes a der Gruppe $(G, \circ)$ ist die kleinste natürliche Zahl k, bei der gilt:

$$a^k = \underbrace{a \circ a \circ \ldots \circ a}_{k \text{ mal}} = 1 \tag{4.46}$$

  Hierbei ist 1 das Identitätselement und muss nicht der Zahl 1 entsprechen.

- Eine Gruppe, welches ein Element α mit der maximalen Ordnung $ord(\alpha) = |G|$ besitzt, wird als zyklisch bezeichnet. Elemente mit der maximalen Ordnung werden auch primitive Elemente oder Generatoren genannt.

4.8.3 Beispiel

Nach all diesen Definitionen sollte man nicht die Anwendung aus den Augen verlieren. Durch dieses Konstrukt wird ausgedrückt, dass es möglich ist, Zahlengeneratoren in gewissen Grenzen zu definieren, die sich ab einer bestimmten Anzahl an Ausführungen auch wiederholen. Hierfür ein Beispiel: Man kann eine Gruppe $(G, +)$ mit $G = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ definieren. Zur Erzeugung neuer Elemente wird der Generator 3 und zum Einhalten der Menge G wird die Modulo 10 Operation benutzt. Dadurch lässt sich die Gruppe folgendermaßen als Zahlengenerator benutzen:

$$
\begin{aligned}
(0+3) \mod 10 &= 3 & \qquad (5+3) \mod 10 &= 8 \\
(3+3) \mod 10 &= 6 & \qquad (8+3) \mod 10 &= 1 \\
(6+3) \mod 10 &= 9 & \qquad (1+3) \mod 10 &= 4 \\
(9+3) \mod 10 &= 2 & \qquad (4+3) \mod 10 &= 7 \\
(2+3) \mod 10 &= 5 & \qquad (7+3) \mod 10 &= 0
\end{aligned}
$$

Bei der Anwendung sieht man, das 0 das Identitätselement und 3 ein Generator der Gruppe ist, da seine $|G|$-fache Anwendung wieder das neutrale Element erzeugt und dabei alle Elemente in G generiert wurden. Dies kann für eine repräsentative Abtastung einer Menge benutzt werden.

Kapitel 5

Beschreibung des Systems

Dieses Kapitel gibt einen Überblick über das in dieser Arbeit entstandene System. Im ersten Abschnitt werden externe Bibliotheken behandelt, während der Zweite den strukturellen Aufbau und der Dritte den prozeduralen Aufbau wiedergibt. Am Ende dieses Kapitels gibt es einen Diskussionsabschnitt, bei dem Vor- und Nachteile besprochen werden.

5.1 Die verwendeten Bibliotheken

Damit die Entwicklung des Systems in akzeptabler Zeit geschehen konnte, wurden externe Bibliotheken verwendet, welche einige Grundkompetenzen für das System mit sich bringen.

5.1.1 *OpenCV*

OpenCV (Open Source Computer Vision Library) ist eine Open-Source Bibliothek, die viele Algorithmen im Bereich Bildverarbeitung bereitstellt [OpenCV, 2017b].
OpenCV ermöglicht diesem System die Benutzung von Videos und Bildern zur weiteren Verarbeitung und Analyse. Dabei werden auch die meisten Algorithmen für die Analyse von dieser Bibliothek bereit gestellt. Die Bibliothek wurde für dieses System in der Version 3.3.1 verwendet.

5.1.2 *Eigen*

Eigen ist eine C++ Vorlagenbibliothek für lineare Algebra und verspricht dabei vielseitig, schnell, verlässlich und elegant zu sein [Eigen, 2017b].
Auf den ersten Blick mag es vielleicht redundant wirken, eine zusätzliche Mathematik-Bibliothek einzubinden, da *OpenCV* einige grundlegende mathematischen Konstrukte wie Matrizen, Vektoren und deren Operatoren und vieles mehr mitbringt. Doch in der Anwendung hat sich herausgestellt, dass manche Funktionen, die für diese Arbeit grundlegend sind, in *OpenCV* nicht verfügbar sind. Ein Beispiel dafür ist die Cholesky-Zerlegung. Außerdem kann es für

zukünftige Optimierungen hilfreich sein, diese Bibliothek eingebunden zu haben. Eigen wird für diese Arbeit in der Version 3.3.4 verwendet.

5.1.3 *Qt*

Qt ist ein plattformübergreifendes Softwareframework mit vorgefertigten UI-Elementen, C++ Bibliotheken und einer komplett integrierten Entwicklungsumgebung mit Werkzeugen [Qt, 2017].
Qt ist in dieser Arbeit für das User-Interface zuständig. Durch die vorgefertigten UI-Komponenten ließ sich das Programm übersichtlich designen und durch die Beliebtheit des Frameworks ist auch die Zukunftsfähigkeit des Systems gewährleistet. Abgesehen vom User-Interface wird *Qt* für den prozeduralen Aufbau verwendet, da dieser die Oberfläche nicht blockieren soll. *Qt* wird für diese Arbeit in der Version 5.8 verwendet.

5.2 Der Aufbau des Systems

Das System wird in Abb. 5.1 als UML-Klassendiagramm wiedergegeben. Dabei ist die UML-Notation zur besseren Lesbarkeit abgewandelt worden.

5.2.1 Beschreibung der Pakete

In diesem Abschnitt wird die Funktionalität der einzelnen Pakete erläutert:

- **GUI**: Alle Klassen in diesem Paket werden für die Darstellung der Anwendung und die Kommunikation zwischen User-Interface und Funktionalität benutzt.
- **UIHandling**: Als Unterpaket der GUI werden hier Skalierungen und Fenstergrößen berechnet. Die enthaltenen Klassen sind somit für das Aussehen der Anwendung verantwortlich.
- **Bibliotheken**: Dieses Paket enthält Klassen, welche wiederkehrende Routinen realisieren, die beinahe im gesamten System verwendet werden.
- **VideoAccess**: Hier sind Klassen enthalten, die es ermöglichen, aus einem Video einzelne Bilder zu extrahieren und diese in der GUI darzustellen.
- **Calibration**: Alle hier enthaltenen Klassen ermöglichen dem System die Anwendung der Kalibrierungsverfahren.
- **Analyzation**: Dieses Paket enthält sämtliche Klassen, die zur Auswertung der Videos benutzt werden.
- **B-Human-Includes**: Ein Paket, welches Klassen aus dem B-Human-Framework kapselt, die für die Sensorintegration verwendet werden.

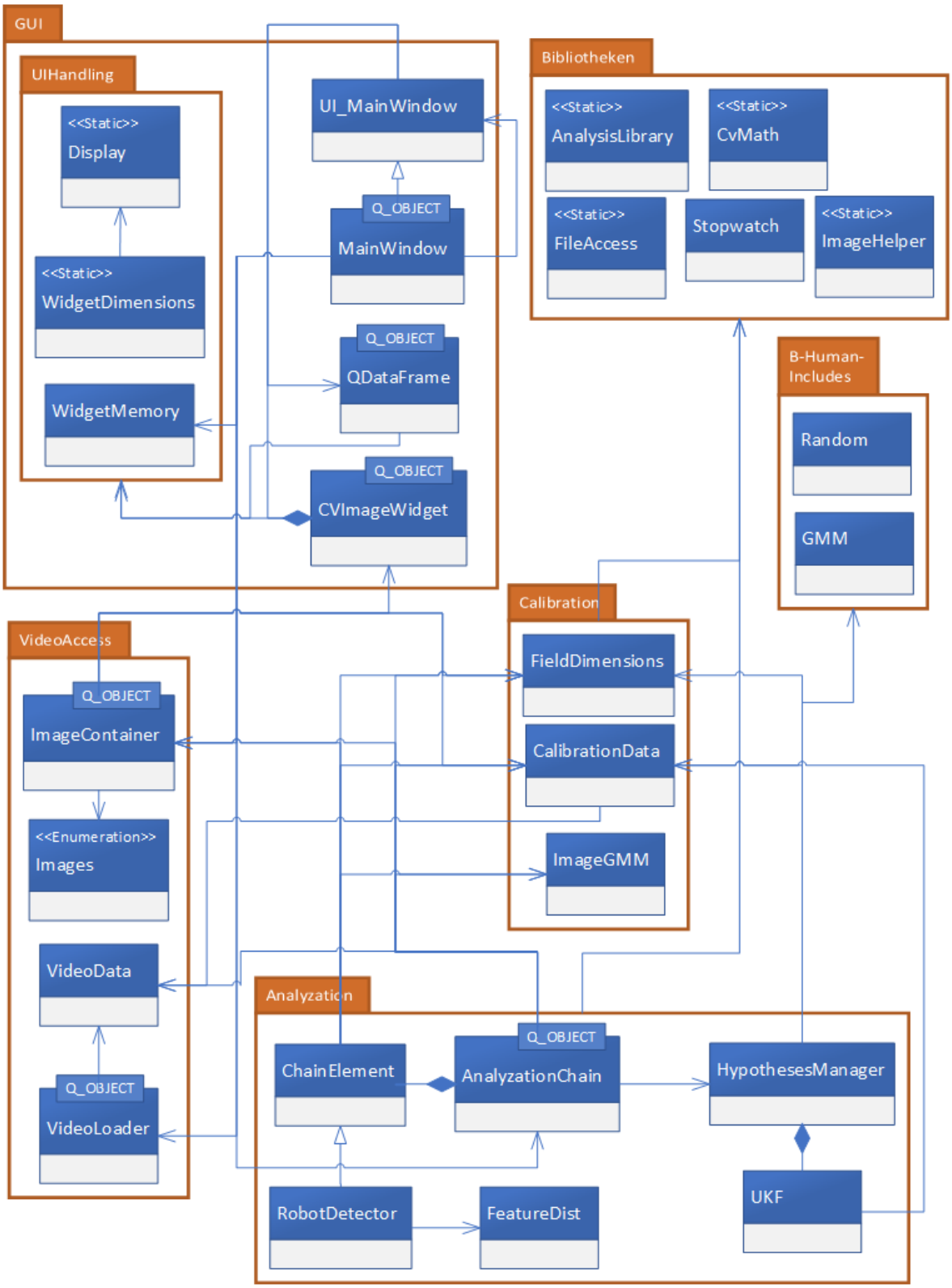


Abbildung 5.1 Das Klassendiagramm zur Beschreibung des Systems. Die UML-Notation wurde um Attribute und Methoden der Klassen reduziert und durch die Verwendung von Vorlagenparametern erweitert. Dadurch bleibt das Diagramm übersichtlich, während Qt-Objekte als solche gekennzeichnet werden.

5.2.2 Beschreibung der Klassen

In diesem Abschnitt wird die Funktionalität jeder Klasse des Systems erläutert, wobei das Paket B-Human-Includes ausgelassen wird, da es aus dem B-Human-Framework entnommen wurde.

- **UI_MainWindow**: Eine aus der UI-Datei generierte Oberklasse, welche zur Anbindung der im *Qt*-Designer entwickelten GUI an die vom Entwickler modifizierbare Klasse MainWindow-Klasse benutzt wird. In dieser Oberklasse werden alle GUI-Elemente mit ihren Eigenschaften aus der UI-Datei deklariert und initialisiert. Ein Zugriff auf GUI-Elemente erfolgen nur über die Instanz von *UI_MainWindow*.
- **MainWindow**: Die Hauptklasse der gesamten GUI, welche die grundlegende Anbindung zur Funktionalität darstellt. In dieser Klasse wird die Kommunikation des *Qt* Slot-Signal-Systems deklariert, was die Funktionalität der Oberfläche, den prozeduralen Aufbau und die weitere Kommunikation der Komponenten ermöglicht.
- **QDataFrame**: Das Oberflächenfenster zur Darstellung von Daten. Über Slots wird eine Liste von Strings entgegen genommen und visualisiert.
- **CVImageWidget**: Das Oberflächenfenster zur Darstellung von Bildern. Es wird ein Bild im *OpenCV*-Format entgegen genommen und in ein in *Qt* darstellbares Format überführt. Die weitere Funktionalität besteht darin, die Nutzereingaben zu verarbeiten, um eine Verschiebung oder eine Skalierung des Bildes zu ermöglichen.
- **Display**: Eine Helferbibliothek, die den aktuellen Bildschirm ermittelt und über die festen Proportionsparameter die Skalierung berechnet.
- **WidgetDimensions**: Eine Helferbibliothek, die *Display* benutzt, um die optimalen Ausmaße von *CVImageWidget* und *QDataFrame* zu berechnen. Sie wird beim Konstruktoraufruf der beiden Widgets aufgerufen.
- **WidgetMemory**: Diese Klasse kommuniziert zwischen den verschiedenen Instanzen von *CVImageWidget*, wie die aktuelle Skalierungsstufe und Zeichenposition aussieht. Die zentrale Instanz wird vom *MainWindow* verwaltet. Die Änderungen des aktiven *CVImageWidget* wird dem *MainWindow* berichtet, welches durch das *WidgetMemory* die anderen Instanzen aktualisiert. Dadurch ist in jeder Ansicht derselbe Ausschnitt zu sehen.
- **AnalysisLibrary**: Eine statische Bibliothek, die viele Funktionen beinhaltet, die bei der Bildanalyse wiederkehren, wie zum Beispiel Wrapper für *OpenCV*-Funktionen wie Houghtransformationen, Konturdetektion und Bilditerationsfunktionen.
- **FileAccess**: Eine Bibliothek, die eine Funktion zum Lesen systemeigener Konfigurationsdateien bereitstellt.

- **Stopwatch**: Eine Klasse, die zur Laufzeitanalyse verwendet werden kann.
- **ImageHelper**: Eine Bibliothek zum Konvertieren von Bildern in *OpenCV* zum *Qt* Format und zurück. Außerdem werden hier auch Funktionen bereitgestellt, um auf Bilder eines Videos zuzugreifen und diese gegebenenfalls zu skalieren. Diese Bibliothek wird nur vom *CVImageWidget* aufgerufen.
- **CvMath**: Eine statische Bibliothek zur Verwendung mathematischer Operationen wie Längenberechnung, Winkelbestimmung und Drehungen auf *OpenCV*-Konstrukte. Dabei sind hier auch Konversionen zu anderen *OpenCV*-Klassen enthalten sowie die Berechnung des repräsentativen rotierten Rechtecks.
- **ImageContainer**: Die Klasse für die zentrale Verwaltung aller Bilder, die aus einem Video gelesen werden. Kantenbilder und weitere Abstraktionsebenen wie die Feldansicht werden hier vorberechnet. Wenn eine intrinsische Kamerakalibrierung vorliegt, wird diese verwendet, um das erhaltene Bild zu entzerren. Dabei werden hier auch berechnete Bildregionen, die durch Kalibrierung und den Fokus aus Kapitel 6 gegeben sind, gespeichert und angewandt.
- **VideoData**: Ein Container aller wichtigen Daten über das Video. Zu diesen Daten gehören der Name der Videodatei, die Anzahl der Bilder, die Auflösung und die Anzahl der Bilder pro Sekunde. Diese Klasse wird vom *VideoLoader* berechnet und wird ausgehend von der *AnalyzationChain* allen Klassen mitgeteilt, die Konfigurationsdateien laden. Dies betrifft in der jetzigen Fassung nur *CalibrationData*.
- **VideoLoader**: Die Klasse, um auf das Video zuzugreifen und einzelne Bilder daraus auszulesen. Sie liefert dabei nicht nur die Video-Player-Funktionalität des Systems, sondern stellt auch *VideoData* bereit und gibt den Lesestatus an.
- **FieldDimensions**: Diese Klasse beinhaltet eine Beschreibung des Spielfeldes in Weltkoordinaten und beschreibt die verschiedenen Kalibrierungmuster als eine Sammlung von Punktlisten und ebenso die Feldlinien. Außerdem wird diese Klasse von *ImageContainer* verwendet, um die Feldansicht zu zeichnen und kann von *ChainElement*s benutzt werden, um zu überprüfen, ob ein Punkt sich auf dem Grün des Feldes befindet.
- **CalibrationData**: Die zentrale Klasse zur Anwendung der Kalibrierungsalgorithmen und Speicherung der dafür benötigten und resultierenden Daten. Diese Klasse enthält sämtliche Parametrisierung der Kalibrierungsverfahren und lädt bereits erstellte Konfigurationen, um die Entzerrung von Bildern und Projektionen zu ermöglichen. Verschiedene Aspekte der Kalibrierung werden in Kapitel 7 beschrieben.
- **ImageGMM**: Diese Klasse ermöglicht eine Farbkalibrierung über ein *Gaussian Mixture Model*, gegeben ein Beispielbild und eine Anzahl der geforderten Klassen. Für die weitere Verwendung werden Funktionen zur Berechnung von segmentierten Bildern bereitgestellt. Die genaue Funktionsweise wird in Kapitel 6.3 beschrieben.

- **ChainElement**: Die Vorlage für alle Klassen, die eine Analyse auf dem Videomaterial durchführen sollen. Den erbenden Klassen wird der Zugang zu Bildern, Farbkalibrierung, Transformationen und dem Feldaufbau ermöglicht.
- **RobotDetector**: Eine Analyseklasse, welche Roboter im Videobild erkennt und deren Fußpunkte als Ausgabe liefert. Die genaue Funktionsweise wird in Kapitel 8 beschrieben.
- **AnalyzationChain**: Die Analysekette, die für jedes Bild im Video durchlaufen wird. Sie verwaltet alle Operationen auf den Bildern. Dabei übernimmt diese Analysekette nicht nur die Aufgabe, den Kettengliedern die benötigten Informationen zukommen zu lassen, sondern führt auch am Anfang des Videos eine eigene Analyse aus, um einen Fokus auf das Spiel zu ermöglichen in dem sie eine Farbkalibrierung durchführt, wie in Kapitel 6.3 beschrieben.
- **HypothesesManager**: Diese Klasse sammelt die Daten aller erfassten Roboter und repräsentiert diese als eine Liste von Schätzungen über ein *Unscented Kalman-Filter* für jeden Roboter. Dabei wird für die spätere Verarbeitung diese Liste auch als *Gaussian Mixture Model* angesehen. Auf Anfrage über GUI erstellt diese Klasse eine CSV-Datei mit den Referenzdaten. Die genaue Funktionsweise wird in Kapitel 9.2 und Folgenden beschrieben.
- **UKF**: Die Realisierung eines *Unscented Kalman-Filters* zur Filterung der geschätzten Roboterposition in Weltkoordinaten, bei denen Bildpositionen als Messung genommen werden. Die konkrete Modellierung wird in Kapitel 9 erläutert.
- **FeatureDist**: Eine Struktur, die es dem *RobotDetector* ermöglicht, Features über ihre Distanz zueinander zu clustern.

Der Aufbau dieses Systems ist experimentell entstanden und nicht in einem durchgeplanten Entwicklungsprozess konstruiert worden, da viel mit den hier verwendeten Technologien experimentiert wurde. Die absoluten Kerneinheiten zum Verständnis des Systems sind: *ImageContainer*, *CalibrationData*, *ImageGMM*, *RobotDetector*, *AnalyzationChain*, *HypothesesManager* und *UKF*. Die Kernaspekte dieser Module werden in nachfolgenden Kapiteln behandelt:

- Kapitel 6 → *AnalyzationChain* und *ImageGMM*
- Kapitel 7 → *CalibrationData*
- Kapitel 8 → *ImageContainer* und *RobotDetector*
- Kapitel 9 → *HypothesesManager* und *UKF*

5.3 Der Prozessaufbau

Die Klasse *MainWindow* sichert die Kommunikation zwischen den Komponenten. Dabei ist es notwendig, die jeweiligen Komponenten als *Q_OBJECT* zu definieren, damit man das Signal-Slot-System von *Qt* verwenden kann. Die Kommunikation zum Durchlaufen der Videoanalyse wird in Abb. 5.2 dargestellt. Diese Hauptroutine ist darauf ausgelegt, eigene Threads zu benutzen, damit der Hauptthread die GUI nicht blockiert. Ein Vorteil der Einführung des Systems ist, dass während der ganzen Videoanalyse nur wenig über Mutex gesichert werden muss, da nur eine Variable einen gesicherten Zugriff braucht. Der Prozess ist so konzipiert, dass dieser, einmal angestoßen, einzeln die Bilder aus dem Video extrahiert, diese in die Vorverarbeitung übergibt und dann die Analyse auf diesem Material durchgeführt wird. Dies geschieht mit jedem Bild im Video.

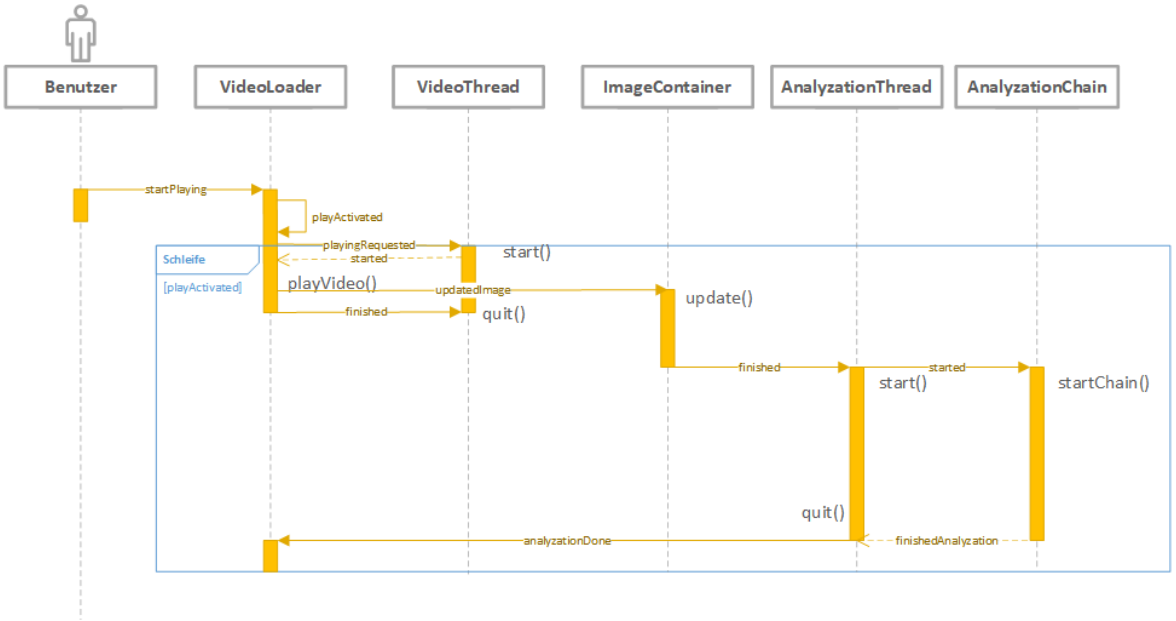


Abbildung 5.2 Der prozedurale Aufbau der Videoanalyse.

Nicht alle Slots und Signale im System lassen sich in Abb. 5.2 wiederfinden, da für Entwicklungs- und Kalibrierungszwecke auch Daten zwischen Klassen kommuniziert werden. Diese Kommunikation nimmt keinen grundlegenden Einfluss auf die Kernanwendung des Systems, sondern stellt nur sicher, dass die vorbereitenden Prozesse für die Analyse ausgeführt werden können und dass die GUI benutzt werden kann.

5.4 Diskussion

Das System wurde mit einem gewissen Effizienzanspruch entwickelt, dennoch ist es möglich, dass die Kommunikation über *Qt* nicht die effizienteste Lösung ist. Auch die Verwendung von *OpenCV* ist ein Kompromiss zwischen einer großen Bandbreite an Werkzeugen und Echtzeit-

fähigkeit, also die Fähigkeit, dieselbe Anzahl der Bilder pro Sekunde zu analysieren wie in der Quelle enthalten sind. Ein deutlicher Schwachpunkt des Systems ist das *CVImageWidget*, welches ein Bild im *OpenCV*-Format nimmt und in ein *QImage* konvertiert, um dieses überhaupt darstellen zu können. Dieser Prozess stellte sich in der Entwicklung als sehr langsam heraus und könnte durch die Benutzung von *OpenGL* eventuell behoben werden. Für weitere Entwicklungen wäre es sinnvoll, eine durchgängige modulare Struktur des Systems zu realisieren wie in dem B-Human-System von Röfer u. a. [2017], bei denen Module Repräsentationen im Sinne von Datenstrukturen voraussetzen und bereitstellen können.

Kapitel 6

Der Fokus auf das Spiel

Dieses Kapitel beschreibt ein Kernkonzept dieser Arbeit, dessen Motivation im nächsten Abschnitt erklärt wird. Mögliche Ansätze der Entwicklung werden im Anschluss diskutiert. Am Ende wird die Implementierung erläutert und ein Ausblick der dadurch entstandenen Möglichkeiten gegeben.

6.1 Motivation dieser Entwicklung

Für viele Erkennungsalgorithmen gilt es einige Herausforderungen zu bewältigen. Die Standardprobleme sind [vgl. Suppa, 2017]:

1. Änderungen des Blickpunktes
2. Beleuchtungsänderungen
3. Verdeckungen
4. Skalierungen
5. Deformierungen
6. Hintergrundstörungen
7. Variationen innerhalb der Detektionsklasse

Die Lösungsansätze für diese Probleme sind sehr unterschiedlich.

1. Der Blickpunkt sollte bei diesem System nicht geändert werden. Dies ist eine Annahme, die hier einfach getroffen werden kann, da man bei den Spielen in der SPL die Kamera einmal aufstellt und dann nicht mehr verändert, weder die Position noch die Ausrichtung. Der Nachteil dieser Annahme liegt darin, dass man dadurch das Prinzip der Kamerasteuerung nicht nutzen kann, wie Misu u. a. [2009] es mit ihrer sekundären

steuerbaren Kamera taten, wobei dieser potentielle Vorteil bei einer Kamera, die nicht von einem Computer gesteuert wird, vernachlässigbar ist. Der eindeutige Vorteil besteht darin, dass so für jedes Video nur einmal die extrinsische Kamerakalibrierung ausgeführt werden muss, wie in Kapitel 7.3 beschrieben.

2. Nach dem Regelwerk [RoboCup Technical Committee, 2017] sollte die Beleuchtung stabil sein. Allerdings ist dies in der Realität häufig nicht gegeben, weshalb die Roboter bei den offiziellen Spielen diverse Probleme in ihrer visuellen Erkennung haben. Im Zuge dessen ist eine Beleuchtungsunabhängigkeit auch für dieses System wünschenswert, damit es auch zukunftsfähig ist.

3. Die Verdeckung wird mit nur einer aufgestellten und statisch positionierten Kamera das größte Problem, welches nur mit weiteren Kameras perfekt gelöst werden kann. Viele Forschungsarbeiten versuchen, dieses Problem mit einer einzelnen Kamera zu lösen. Hier kann man sich lediglich bei dem Aufbau des Erkennungsalgorithmus überlegen, wie dort eine Verdeckung die Erkennung beeinflussen könnte und wie man dann eine gute Messung erzielt. Notwendigerweise muss man gelegentlich in Kauf nehmen, dass eine Messung nicht stattfindet, weshalb die Modellierung von Robotern mit Kalman-Filtern angemessen ist.

4. Die Skalierung muss sich auch in der Robotererkennung wiederfinden. Das Skalierungsproblem ist ebenfalls in dieser Aufgabenstellung vorhanden, da ein Roboter auf der einen Seite des Feldes wesentlich größer ist als auf der anderen. Deshalb darf nicht mit festen Größeneigenschaften gearbeitet werden, sondern mit Größenverhältnissen.

5. Deformierungen sind gegeben, da die verwendete Kamera eine *GoPro* ist. Diese Kameraart besitzt starke Verzerrungen, weshalb klassische Bildverarbeitungsalgorithmen wie Hough-Transformationen, die geometrische Formen erkennen, nicht ohne eine vorherige Kalibrierung sinnvoll benutzt werden können. Die Kalibrierung wird in Kapitel 7 erläutert.

6. Hintergrundstörungen sind bei einer weitwinkligen Kamera wie der *GoPro* eindeutig gegeben. Das Feld nimmt zwar, bei guter Positionierung der Kamera, einen Großteil des Bildes ein, aber man sieht auch sehr viel im Hintergrund. Deswegen darf kein einfacher Feature-Erkenner auf das Gesamtbild angewendet werden und eine reine Bearbeitung des Feldausschnittes wäre wünschenswert. Ein Beispiel einer groben Feature-Detektion auf dem gesamten Bild ist in Abb. 6.1 zu sehen.

7. Variationen innerhalb der Detektionsklasse sind nahezu ausgeschlossen, da alle Teams den *NAO* benutzen. Dies sichert eine allgemeine Erscheinungsform, aber kann sich in den verschiedenen Farbvariationen in der Auslieferung sowie in den Trikots der jeweiligen Teams und anderweitigen Verzierungen nach dem Regelwerk [RoboCup Technical Committee, 2017] unterscheiden.

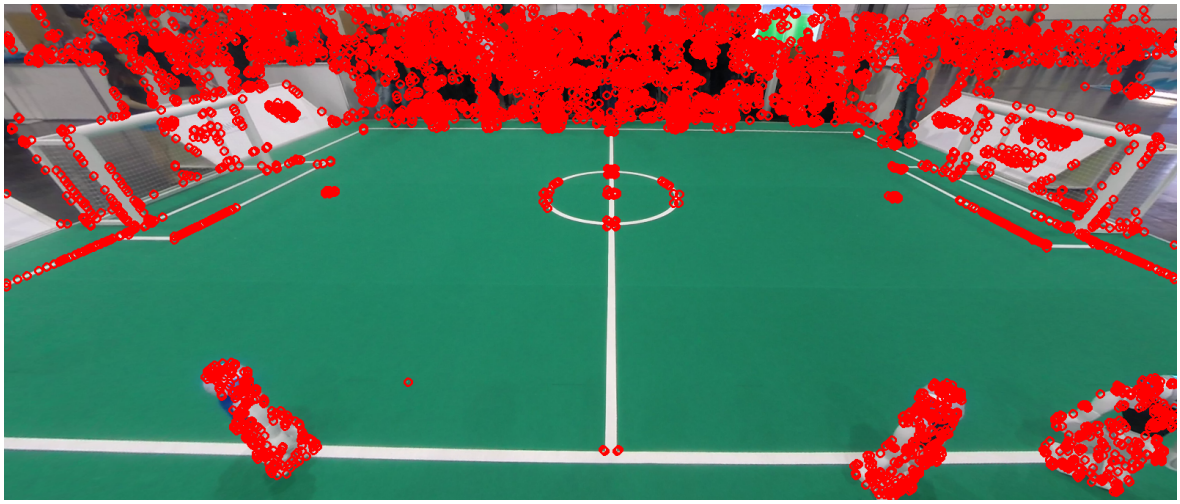


Abbildung 6.1 Anwendung von FAST auf das gesamte Bild.

6.2 Diskussion verschiedener Ansätze

Anhand der Hintergrundstörungen in Abb. 6.1 ist ersichtlich, dass ein Zuschneiden des realen Bildes zu einem Region of Interest (kurz ROI), eine Bezeichnung für ein Teilbild, in dem eine Untersuchung statt finden soll, wünschenswert ist. Ein weiterer Vorteil, der durch eine Verkleinerung des Bildes entsteht, ist die geringere Rechenzeit, die man für die Auswertung eines Bildes benötigt, da die meisten Bildverarbeitungsalgorithmen im Aufwand zur Anzahl der Pixel skalieren. Bei einem statischen Blickpunkt ist es auch möglich, den ROI nur einmal zu berechnen und dann für jedes kommende Bild wieder zu verwenden. Um aber den ROI zu berechnen, bieten sich verschiedene Möglichkeiten:

1. Berechnung über geometrische Formen: Das Feld hat eine rechteckige Form und lässt sich damit aus vier Linien zusammensetzen. Dabei stößt man auf mehrere Probleme. Wie zuvor erwähnt, ist die Extraktion von geometrischen Formen in verzerrten Bildern absolut nicht trivial. Dies liegt an der Suche nach geraden Linien, die in verzerrten Bildern keine geraden Linien sind, was dem Prinzip der Transformation der Lochkamera widerspricht, in dem transformierte Linien auch Linien bleiben. Dadurch wäre eine qualitative hochwertige Entzerrung des Bildes eine Voraussetzung für die weitere Analyse. Ergänzend dazu ergeben sich durch den Aufbau des Feldes viele andere Linien, die detektiert werden könnten. Ebenso ist es möglich, dass die Linien durch Roboter oder Schiedsrichter, die auf den Linien stehen, unterbrochen sind. Diese Gründe machten diesen Ansatz in der Implementierungsphase nicht allgemein praktikabel.

2. Berechnung über Farben: Eine Berechnung über Farben ist insofern anwendbar, dass die Farbe des Feldes als grün definiert ist. Da mit dem *OpenCV*-Framework ein Bild leicht in den HSV-Farbraum, in Kapitel 4.1.2 erwähnt, konvertiert werden kann und dann mit einiger Präzision auf Farben untersucht werden kann, lohnt sich eine genauere Betrachtung des Ansatzes. Im gesamten Bild ist nicht in jedem Video Grün die häufigste

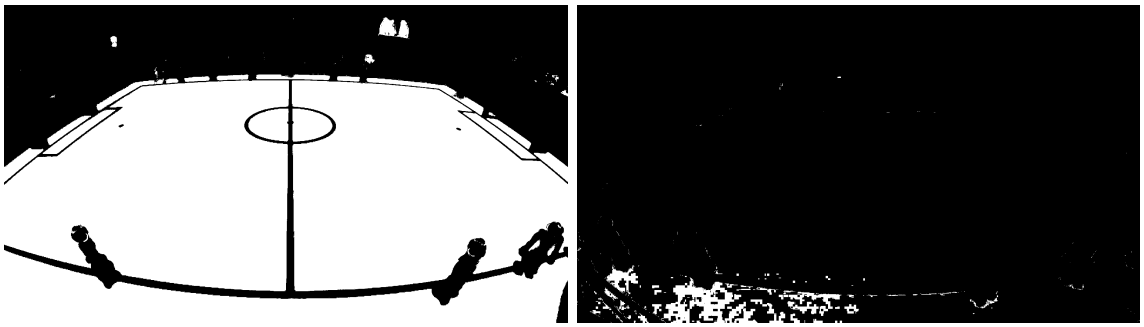
Farbe. Daher muss man nicht nur eine Werte-Häufigkeit analysieren, sondern auch die genauen Werte und ihre Korrelation zueinander. Daher die folgenden Möglichkeiten:

- Feste Schwellwerte: Eine einfache Herangehensweise der farbbasierten Extraktion ist eine Binarisierung über die *inRange* Funktion, die eine obere und untere Schranke entgegen nimmt, um ein Binärbild zu erstellen. Dabei entspricht in diesem Bild jedes weiße Pixel einem Ursprungspixel, welches zwischen den Werten liegt. Die Schwäche dieses Ansatzes sind die festen Schwellwerte, die keine Beleuchtungsunabhängigkeit garantieren können, welche aber wie an Abb. 6.2 verdeutlicht dringend benötigt wird. Wenn man dann einen festen geeigneten Schwellwert für Grün auf Basis der Indoor Competition 2016 nimmt, so lässt dieser sich nicht auf die Outdoor Competition anwenden, wie in der Binarisierung in Abb. 6.3 dargestellt.



(a) Outdoor Competition 2016 (b) Indoor Competition 2016 (c) Indoor Competition 2017

Abbildung 6.2 Ein Überblick der verschiedenen Beleuchtungsbedingungen auf Wettkämpfen in Kombination mit unterschiedlichen Teppichen.



(a) Binarisierung Indoor (b) Binarisierung Outdoor

Abbildung 6.3 Die Binarisierung mit festen Schwellwerten.

- *K-Means*: Der nächste logische Schritt wäre es, einen Ansatz mit adaptiven Schwellwerten auszuprobieren. Wenn man dann eine Liste aus allen Pixelwerten in eine Anzahl von k Clustern aufteilt mit dem in *OpenCV* implementierten *K-Means*-Algorithmus, so lässt sich auf ein bereits auf die untere Hälfte zu geschnittenes Bild der grüne Cluster über die Anzahl der jeweiligen Cluster-Pixel im Bild ermitteln. Über diesen Cluster lassen sich ein maximaler und ein minimaler Wert für Grün finden. Selbst dieser adaptive Ansatz war unzureichend, da in der Outdoor Szene

große Lücken entstehen und daher nur eine Annäherung entsteht, wie in Abb. 6.4 dargestellt.



Abbildung 6.4 Binarisierung über K-Means in der Outdoor Competition.

- *Gaussian Mixture Model*: Eine adaptive Variante, die häufig zur Klassifizierung von Hintergrund und Vordergrund benutzt wird, ist das *Gaussian Mixture Model*. Sie dienen als unüberwachtes Lernverfahren, welches ebenfalls ein Clustering mit k Clustern anstrebt. Wichtig ist dann die Interpretation des Lernergebnisses. Dieses Verfahren lieferte in der Entwicklung des Systems die besten Ergebnisse.

6.3 Farbkalibrierung über *Gaussian Mixture Model*

Die Implementierung eines *Gaussian Mixture Model* für Lernverfahren findet sich in *OpenCV* im Namespace *ml* in der Klasse *EM*. Der Algorithmus versucht nach OpenCV [2016], in der Notation von Kapitel 4.4.1, ein *Gaussian Mixture Model* in einem d dimensionalen Raum über m Gauß-Verteilungen der Form

$$p(x, \mu_k, \Sigma_k, \pi_k) = \sum_{k=1}^{m} \pi_k \, p_k(x) \qquad \pi_k > 0, \sum_{k=1}^{m} \pi_k = 1 \tag{6.1}$$

$$p_k(x) = \varphi(x, \mu_k, \Sigma_k) \tag{6.2}$$

$$= \frac{1}{(2\pi)^{\frac{d}{2}} |\Sigma_k|^{\frac{1}{2}}} \exp\left\{ -\frac{1}{2} (x - \mu_k)^T \, \Sigma_k^{-1} \, (x - \mu_k) \right\} \tag{6.3}$$

über alle Parameter μ_k, Σ_k und π_k zu schätzen. Diese Parameter werden über *Maximum-Likelihood Estimation* auf einer Trainingsmenge von Beispielvektoren $\{x_1, x_2, \ldots, x_n\}$ ermittelt. Diese Parameter sollen das Folgende erfüllen:

$$L(x, \theta) = \log p(x, \theta) = \sum_{i=1}^{N} \log \left(\sum_{k=1}^{m} \pi_k p_k(x) \right) \to \max_{\theta \in \Theta} \tag{6.4}$$

$$\Theta = \left\{ (\mu_k, \Sigma_k, \pi_k) : \mu_k \in \mathbb{R}^d, \Sigma_k = \Sigma_k^T > 0, \Sigma_k \in \mathbb{R}^{d \times d}, \pi_k \geq 0, \sum_{k=1}^{m} \pi_k = 1 \right\} \tag{6.5}$$

Der Algorithmus zur Parameteranpassung arbeitet in zwei Schritten. Der erste Schritt ist der Expectation-Schritt, bei dem die Wahrscheinlichkeit für das i-te Sample x_i, mit $i = 1, \ldots, n$, für jede der Verteilungen berechnet wird, also die Funktion p_{ki} der folgenden Form ausgewertet wird:

$$p_{ki} = \frac{\pi_k \, \varphi(x, \mu_k, \Sigma_k)}{\sum_{j=1}^{m} \pi_j \, \varphi(x, \mu_j, \Sigma_j)} \tag{6.6}$$

Anschließend werden die Parameter im Maximization-Schritt über die berechneten Wahrscheinlichkeiten angepasst:

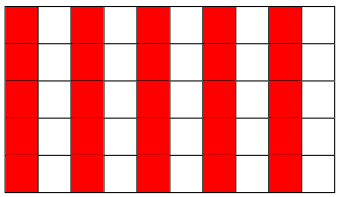
$$\pi_k = \frac{1}{n} \sum_{i=1}^{n} p_{ki} \tag{6.7}$$

$$\mu_k = \frac{\sum_{i=1}^{n} p_{ki} \, x_i}{\sum_{i=1}^{n} p_{ki}} \tag{6.8}$$

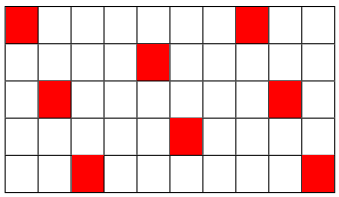
$$\Sigma_k = \frac{\sum_{i=1}^{n} p_{ki} \, (x_i - \mu_k)(x_i - \mu_k)^T}{\sum_{i=1}^{n} p_{ki}} \tag{6.9}$$

Nach dieser Erklärung des zugrunde liegenden Algorithmus wird offensichtlich, dass je nach Problemstellung der Parameterraum sehr groß werden und sehr viel Rechenzeit zur Lösung des Problems notwendig werden kann. Die Eingabe für die Klassifizierung in diesem System ist ein Bild im HSV-Format. Dies bedeutet, dass der Parameter $d = 3$ ist. Der gewählte Farbraum ist zur Differenzierung von Farben besser geeignet als der RGB-Farbraum, da er die Beleuchtung besser berücksichtigt.
Der nächste wichtige Parameter ist k, welcher die Anzahl der Gauß-Verteilungen (also Cluster) angibt und somit auch grundlegend über die notwendige Rechenzeit entscheidet. Bei Versuchen erzielte $k = 4$ gute Resultate. Eine Begründung, warum man in vier Farben unterscheiden sollte, ist dadurch gegeben, dass es prinzipiell genau so viele Farben gibt, die auf dem Spielfeld vorhanden sein sollten. Dies beinhaltet das Grün des Feldes, das Weiß der Linien und der Roboter und die zwei unterschiedlichen Teamfarben.
Der letzte Parameter, der deutlich das Anlernen des *Gaussian Mixture Model* beeinflusst, ist n, welcher die Anzahl der Samples in der Trainingsmenge beschreibt. Prinzipiell will man möglichst repräsentative Trainingsdaten haben und kann dafür ein *Gaussian Mixture Model* über das gesamte Bild lernen. Wenn man dies aber bei Aufnahmen einer *GoPro* anwendet, so liegt man bei einer „niedrigeren“ Auflösung der Testvideos bei $n = 1920 \cdot 1080 = 2073600$ und bei der „höheren“ bei $n = 2704 \cdot 1524 = 4120896$. Diese Anzahl an Trainingsdaten macht das System nahezu unnutzbar. Der Parameter muss für die weitere Anwendbarkeit deutlich verringert werden. Um dies zu erreichen ist der logische Schluss, jeden x-ten Pixel für das Training heranzuziehen. Um dabei eine repräsentative Menge zu erhalten, bedient man sich des Prinzips der zyklischen Gruppen, die in Kapitel 4.8 formal eingeführt wurden. Die

Herausforderung besteht darin, die Gruppe so zu wählen, dass das entstehende Muster sich nicht zu häufig wiederholt und dabei die Trainingsmenge nicht zu groß wird. Ein Beispiel für die Anwendung solcher Gruppen auf Bilder mit ihren resultierenden Mustern ist in Tab. 6.1 zu sehen. Um einen guten Kompromiss zwischen diesen Anforderungen zu finden, haben sich Generatoren α als geeignet herausgestellt, die die Bedingungen $\frac{|G|}{3} < \alpha < |G| \cdot \frac{3}{4}$ und $\gcd(\alpha, |G|) = 1$ erfüllen. Dabei ist $|G|$ die Bildbreite, die zum Zeitpunkt der Farbanalyse der Breite des entzerrten Bildes entspricht, was bei der höheren Auflösung einer Breite von 2151 Pixeln entspricht. In dem System wird $\alpha = 733$ verwendet, da bei beiden Bildformaten die Anwendung dieser Analyse in der Release-Konfiguration nahezu echtzeitfähig ist und die Farben trotz schwieriger Versuchsaufbauten repräsentativ sind.



Anwendung der Gruppe mit Generator 2.



Anwendung der Gruppe mit Generator 7.

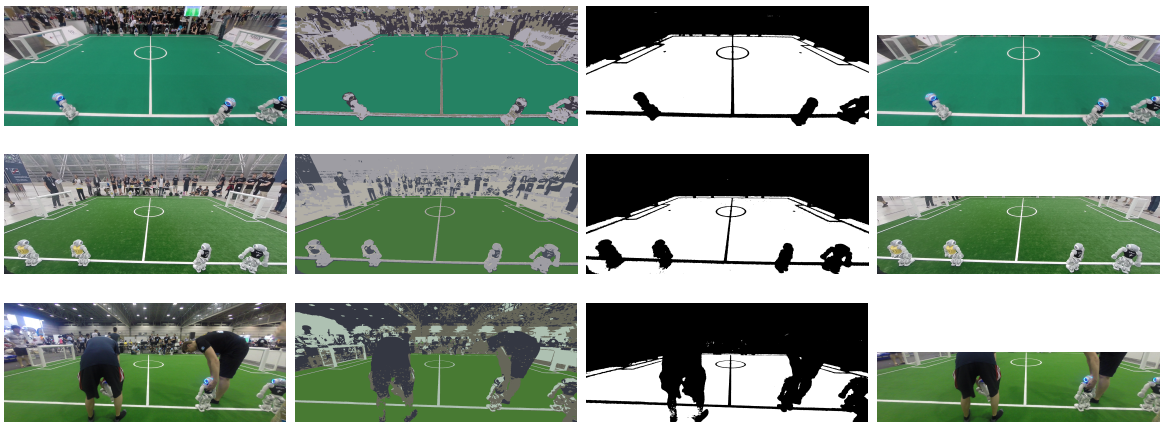
Tabelle 6.1 Beispielhafte Illustration der Anwendung von Gruppen der Basis $(\{0, \ldots, 9\}, +)$ auf Bildern.

Wenn über die Trainingsmenge dann das *Gaussian Mixture Model* angelernt wurde, so kann man, wie in Formel 4.13 gezeigt, alle Pixel im Bild mit einer Verteilung markieren und damit eine Segmentierung erreichen. Dabei gilt ein Pixel einer Verteilung zugehörig, wenn die normierte Wahrscheinlichkeit, dass der Pixel dieser Verteilung angehört, größer ist als 0.98. Dieser Ansatz erwies sich, wie in Abb. 6.5 dargestellt, als auf mehreren Testvideos anwendbar. Wie die Ergebnisse von (b) bis (d) genau entstehen, wird im nachfolgenden Abschnitt erklärt.

6.3.1 Dadurch entstandene Möglichkeiten

Das Anlernen von Pixelklassifizierungen über ein *Gaussian Mixture Model* bietet einem in der nachfolgenden Verarbeitung viele Möglichkeiten. Jede der vier angelernten Klassen wird über einen Mean μ_k und eine Kovarianzmatrix Σ_k repräsentiert. Wenn man die angelernten Verteilungen überprüfen möchte, so kann man das Bild segmentieren, in dem man jeden Pixel klassifizieren lässt und ihn durch seinen Klassenrepräsentanten μ_k ersetzt. Durch dieses Verfahren sind die Bilder in Abb. 6.5 (b) entstanden. Wenn man dieses Ergebnis untersucht, stellt man fest, dass sich die durch das Anlernen gewonnenen Farbklassen nicht wie erwartet entwickelt haben. Die resultierenden Farbklassen sind:

- Weiß: Wird in einem Großteil der Linien und der Roboter gesehen.
- Grün: Die Farbe des Feldes, welche nur in seltenen Fällen außerhalb gesehen wird.



(a) (b) (c) (d)

Abbildung 6.5 Anwendung der *Gaussian Mixture Model* basierten Region of Interest Extraktion aus verschiedenen Testvideos. In der Spalte (a) sind die Ausgangsbilder, in (b) sind die Mean-Segmentierten-Bilder, in (c) ein Binärbild, bei dem als Grün klassifizierte Pixel weiß sind, und in (d) die extrahierte Bildregion des Feldes.

- Dunkelgrau - Schwarz: Umfasst Kleidung, dabei sind sowohl Roboter-Jerseys als auch menschliche Kleidung vertreten.
- Hellgrau: Eine Farbe, die nach Beobachtung als alles “Sonstige“ bezeichnet werden kann, da sie die Gelenke der Roboter, die Hautfarbe und unzureichend beleuchtetes Weiß umfasst.

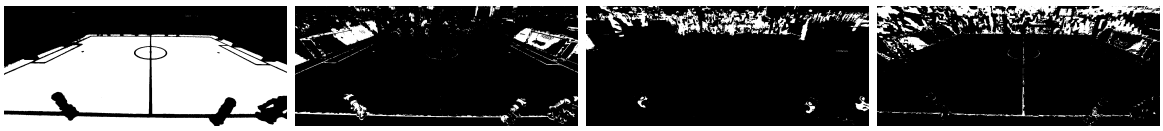
Diese Klassen lassen sich in jedem Video finden und enthalten für weitere Analysen wichtige Informationen. Ein Problem, welches durch dieses unüberwachte Lernverfahren entsteht, ist die Zuweisung, um eine der Verteilungen eindeutig als eine Farbe zu bezeichnen oder auch als solche zu labeln/markieren. Zur Identifizierung der Klassen kann man aber Referenzfarben definieren, die mit den verschiedenen μ_k verglichen werden. Die Farbe, die dann den geringsten Unterschied mit einem μ_k hat, wird als Bezeichner für diese Farbklasse verwendet. Die Referenzfarben für dieses System sind in Abb. 6.6 zu sehen.



(a) Grün $(120°, 50\%, 25\%)$ (b) Weiß $(180°, 0\%, 100\%)$ (c) Schwarz $(180°, 0\%, 0\%)$

Abbildung 6.6 Illustration der verwendeten Referenzfarben im HSV-Farbraum.

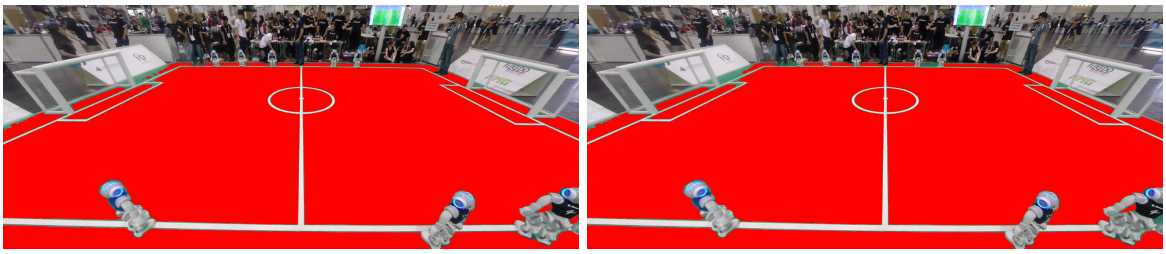
Die Farbklasse, die keiner anderen zugeordnet werden kann, wird als Grau gekennzeichnet. Wenn die Farbklassifizierung durch diese Zuweisung vervollständigt wurde, lässt sich für jede Farbe ein Binärbild erstellen, in dem die jeweilige Klasse als Weiß dargestellt ist. Dadurch lässt sich jede Szene in die vier Farben zerlegen, wie in Abb. 6.7 dargestellt.
Für die Berechnung der Spielfeld-Region werden die grünen Pixel in Form eines Binärbildes



(a) Grün (b) Weiß (c) Schwarz (d) Grau

Abbildung 6.7 Darstellung der möglichen Binärbilder auf Basis der Farbklassen.

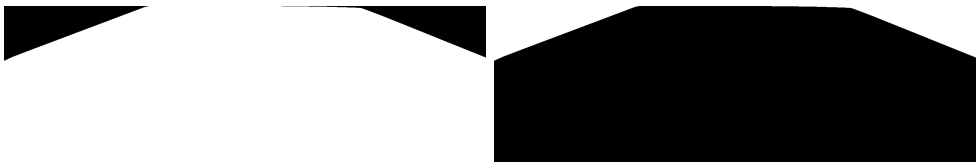
extrahiert, wie in Abb. 6.7 (c). Auf diesem Binärbild werden dann Konturen detektiert, welche in Abb. 6.8 (a) eingezeichnet sind. Aus diesen Konturen werden diejenigen, die von der Fläche her unter einem von der Auflösung abhängigen Schwellwert liegen, gelöscht. Dadurch werden unwesentliche Flächen, die meistens nicht zum Feld gehören, gelöscht. Das Ergebnis findet sich in Abb. 6.8 (b).



(a) Vollständig (b) Selektiert

Abbildung 6.8 Die Konturen, die für die Feldberechnung verwendet werden.

Über die gefilterte Menge der Konturen wird eine Liste von Punkten berechnet, die alle Punkte der Konturen enthält. Diese Liste wird benutzt, um mit der *OpenCV*-Funktion *boundingRect* einen Region of Interest zu berechnen. Diese Berechnung liefert dann Ergebnisse wie in Abb. 6.5 (d) dargestellt. Dieses Bild kann zur weiteren Verarbeitung verwendet werden, da Hintergrundstörungen weitestgehend ausgeschlossen sind. In den in Abb. 6.5 dargestellten Beispielen sind durch eine ähnliche Perspektive immer noch Ausschnitte ohne Feld zu sehen. Diese Regionen können in der Detektion trotzdem zu falschen Detektionen führen, da dort mehrere Farbkonstellationen und andere Merkmale denkbar sind. Um dies zu vermeiden wäre es wünschenswert, wenn auch diese Regionen nicht mehr analysiert werden oder gar verbessernde Eigenschaften für die Detektion besitzen. Durch die bekannte Feldregion und bekannten Farben kann man die Regionen außerhalb des Feldes mit der Feldfarbe einfärben. Die Identifizierung von innerhalb und außerhalb der Feldregion kann man treffen, indem man die konvexe Hülle des Feldes berechnet und in ein separates gleichgroßes Bild zeichnet, wie in Abb. 6.9 (a) dargestellt. Das Resultat lässt sich dann invertieren, wodurch die Hintergrundregionen weiß dargestellt werden, wie in Abb. 6.9 (b) illustriert.
Auf dem invertierten Bild kann man nun eine Konturdetektion durchführen, um eine präzise Repräsentation der Hintergrundregionen zu erhalten. Diese Konturen werden mit der RGB-Farbrepräsentation des Mittelpunktes des grünen Farbclusters dem *ImageContainer* übergeben. Der *ImageContainer* wird dann die Vorverarbeitungskette anstoßen, in der der



(a) Konvexe Hülle (b) Invertierte Hülle

Abbildung 6.9 Die Zwischenschritte der Berechnung der Hintergrundregionen.

berechnete Region of Interest angewendet wird und die ermittelten Regionen dann in dem zuvor ermittelten Grünton eingezeichnet werden. Dadurch ergibt sich, wie in Abb. 6.10 abgebildet, eine vorgefertigte Szene.



Abbildung 6.10 Das Resultat der Vorverarbeitung.

Die in diesem Kapitel vorgestellte Vorverarbeitung ermöglicht eine bessere Analyse des Videos, da jetzt eine Farbklassifikation von Pixeln möglich ist, die nicht auf festen Schwellwerten basiert. Ebenso ist ein Fokus auf den für die Analyse interessanten Teil gegeben, was die Laufzeit verringert und die Anzahl möglicher Fehler deutlich reduziert. Die Vorverarbeitung wird mit ROI-Detektion und Farbanalyse initial direkt nach dem Öffnen des Videos durchgeführt. Um eine robuste Farbklassifizierung gegenüber schwierigen Beleuchtungsbedingungen zu gewährleisten, müsste regelmäßig eine erneute Farbanalyse über das *Gaussian Mixture Model* statt finden, aber in der regulären Systemausführung wird für eine bessere Laufzeit darauf verzichtet. Wie robust die initiale Farbkalibrierung für mögliche farbbasierte Detektoren ist, muss sich in der Evaluation in Kapitel 10 zeigen, wobei in Kapitel 10.5 die initiale Farbkalibrierung mit der wiederholten Kalibrierung bei schwierigen Lichtverhältnissen verglichen wird.

Kapitel 7

Kalibrierung der *GoPro*

Zur Generierung von Referenzdaten und für die Ausführung einer präzisen Analyse ist eine vollständige Kalibrierung der verwendeten Kamera nötig. Die dafür interessanten Aspekte werden in den nachfolgenden Kapiteln behandelt.

7.1 Diskussion der Kalibrierung

Die verwendete Kamera ist eine *GoPro*, welche eine starke Verzerrung, wie in Abb. 7.1 dargestellt besitzt. Um diese Verzerrung für eine perfektes Kamerabild zu entfernen, gibt es mehrere Kameramodelle.



Abbildung 7.1 Eine unbearbeitete Beispielaufnahme der *GoPro*.

Unter der Verwendung von *OpenCV* besitzt man als grundlegendes Kameramodell die Loch-

kamera, deren zugrunde liegende Geometrie in Kapitel 4.3 beschrieben wurde. Doch das allgemeine Modell geht von keinerlei Verzerrung aus. Jedoch gibt es eine Erweiterung, um einfache Verzerrungen zu korrigieren. Man unterscheidet drei in Abb. 7.2 dargestellte Verzerrungsarten. Dabei lassen sich die ersten beiden Arten gut über die Erweiterung korrigieren, während für die dritte das Fish-Eye-Kameramodell verwendet werden muss.



(a) Barrel (b) Pincushion (c) Fisheye

Abbildung 7.2 Beispiele für die drei Verzerrungsarten, entnommen aus [Szeliski, 2010, S.59].

Obwohl ein vollständiges Fish-Eye-Kameramodell wünschenswert wäre, ist es in der verwendeten *OpenCV*-Version nicht anwendbar gewesen. Daher wird das *OpenCV* Lochkamera-Modell von OpenCV [2017a] mit vollständiger Parameterstärke verwendet, welches ein Gleichungssystem aus den folgenden Komponenten enthält:

- $\begin{pmatrix} X \\ Y \\ Z \end{pmatrix}$: Die dreidimensionalen Weltkoordinaten eines Punktes, der projiziert werden soll.
- $\begin{pmatrix} u \\ v \end{pmatrix}$: Die Bildkoordinaten der Projektion in Pixeln.
- $A = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$: Ist die Kameramatrix, auch intrinsische Parameter genannt.
- $\begin{pmatrix} c_x \\ c_y \end{pmatrix}$: Ist der Bildmittelpunkt, der von der optischen Achse durchlaufen wird.
- f_x, f_y: Sind die Brennweiten in Pixeln angegeben.
- R: Ist die Rotationsmatrix, die für die Transformation von der Weltebene zum Kamerakoordinatensystem benutzt wird.
- t: Ist der Translationsvektor, der für die Transformation von der Weltebene zum Kamerakoordinatensystem benutzt wird.

- $(k_1, k_2, p_1, p_2[, k_3, [, k_4, k_5, k_6[, s_1, s_2, s_3, s_4[, \tau_x, \tau_y]]]])$: Die möglichen Verzerrungskoeffizienten des Modells.
  - $k_1, \ldots, k_6$ sind radiale Verzerrungsparameter.
  - p_1, p_2 sind tangentiale Verzerrungsparameter.
  - $s_1, \ldots, s_6$ sind Verzerrungsparameter für ein dünnes Prisma.
  - τ_x, τ_y sind Winkelparameter.

Zur Visualisierung eines Großteils dieser Parameter ist in Abb. 7.3 die *OpenCV* Lochkamera-Skizze zu sehen.

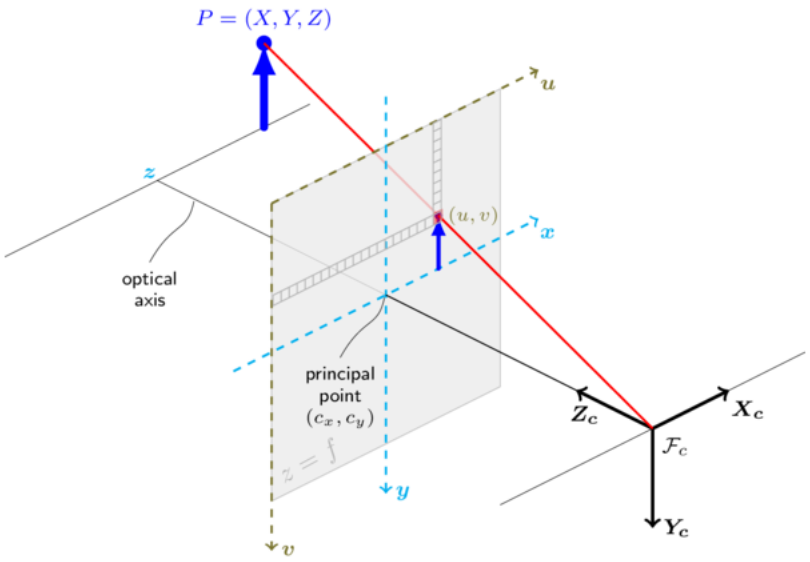


Abbildung 7.3 Darstellung der Lochkamera nach *OpenCV*, entnommen aus [OpenCV, 2017a].

7.2 Intrinsische Kamerakalibrierung

Die intrinsischen Parameter bieten den grundlegenden Vorteil, dass sie nur einmal für eine Kamera auf einem bestimmten Bildformat durchgeführt werden muss und danach auf allen Videos mit diesem Format funktioniert. Vielmehr lassen sich diese Parameter in skalierter Form auch auf die anderen Formate derselben Kamera anwenden. Die Funktion in *OpenCV*, um diese Parameter in diesem System zu bestimmen, ist *calibrateCamera*. Diese Funktion bestimmt nicht nur die intrinsischen Parameter, sondern alle oben genannten. Um eine Kalibrierung hiermit zu erreichen, werden Welt-Bild-Korrespondenzen (Welt- und Bildkoordinaten, die den selben Punkt beschreiben) benötigt, welche in diesem System über eine manuelle Kalibrierung erlangt werden. Allgemein werden für Kamerakalibrierungen wohldefinierte Muster verwendet, die in Bildern automatisch detektiert werden können und deren Ausmaße bekannt sind, weshalb Schachbretter für diese Verfahren geeignet sind. Da dieses

System bereits aufgenommene Videos analysieren soll, kann man ein Muster zur Kalibrierung benutzen, welches auf allen Videos vorhanden sein sollte, nämlich das Spielfeld. Nach den Regeln [RoboCup Technical Committee, 2017] besitzt das Feld bekannte Ausmaße, die zum Aufspannen des Weltkoordinatensystems, wie in Kapitel 4.2.1 beschrieben, benutzt werden. Die integrierte intrinsische Kamerakalibrierung erstellt über das *CVImageWidget* eine Liste von Bildpunkten, deren Reihenfolge wie in Abb. 7.4 auf dem Feld festgelegt ist. Die dadurch generierte Bildpunktliste wird zusammen mit der korrespondierenden Weltpunktliste der Kalibrierung übergeben. Durch dieses Muster sind einige Linien gegeben, welche mit einer geeigneten *OpenCV*-Parametrisierung die Nutzung aller Verzerrungskoeffizienten erzwingen und damit einen für die Entzerrung brauchbaren Datensatz ermöglichen.

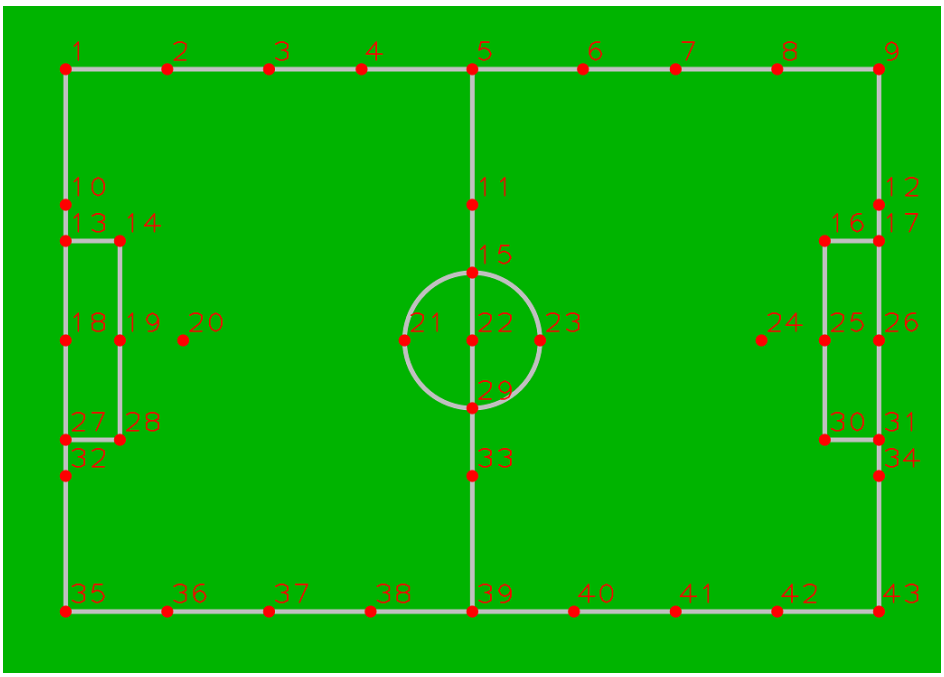


Abbildung 7.4 Das intrinsische Kalibrierungsmuster.

Der hieraus resultierende Datensatz kann direkt mit dem Entzerrungsprozess beginnen, indem wie in Abb. 7.5 das Ausgangsbild (a) mit *undistort* in die Form von (b) gebracht wird. (b) ist deutlich entzerrt, aber besitzt schwarze Ränder, da nicht jeder Pixel sinnvoll in einem entzerrten Bild abgebildet werden kann. Ein Bild mit unregelmäßigem schwarzen Rand ist für die Bildverarbeitung nicht besonders hilfreich, weshalb das Entfernen des Randes wünschenswert ist. In *OpenCV* gibt es dafür die Funktion *getOptimalNewCameraMatrix*, welche die neuen Parameter der Kalibrierung für das entzerrte Bild berechnet und gleichzeitig einen Region of Interest für ein Bild ohne schwarzen Rand ausgibt. Dessen Anwendung auf das entzerrten Bild ergibt das Bild in Abb. 7.5 (c), welches für weitere Verarbeitungszwecke geeignet ist. Obwohl eine intrinsische Kalibrierung vorliegt, mit der einige Analysen durchgeführt werden könnten, sind die Parameter weit von dem Ideal entfernt. Das resultierende Bild besitzt noch genügend Verzerrung, um eine Formdetektion zu erschweren. Hinzu kommt, dass durch das Abschätzen der Punkte, die nicht auf Kreuzungen liegen, eine Fehlerquelle vorliegt, die

zu Parametern führen kann, die keinerlei Präzision in den Projektionen gewährleisten können. Es ist für eine intrinsische Kalibrierung auch unüblich, ein Muster nur in einem Bild zu detektieren und es nur in einer Lage für die Kalibrierung zu verwenden. Damit ist dieses Kalibrierungsverfahren eine Notlösung, wenn keinerlei Möglichkeiten zur weiteren Kalibrierung bestehen.



(a) Verzerrt (b) Entzerrt (c) Zugeschnitten

Abbildung 7.5 Die drei Phasen der Entzerrung eines Bildes.

Das Ziel dieser Arbeit ist es, möglichst gute Referenzdaten zu generieren, weshalb eine qualitativ hochwertige intrinsische Kalibrierung wünschenswert ist. Zu diesem Zweck wurde eine Kalibrierung mit einem Schachbrett unter Laborbedingungen bei einer Auflösung von 2704×1524 durchgeführt. Eine Beispielaufnahme ist in Abb. 7.6 zu sehen.

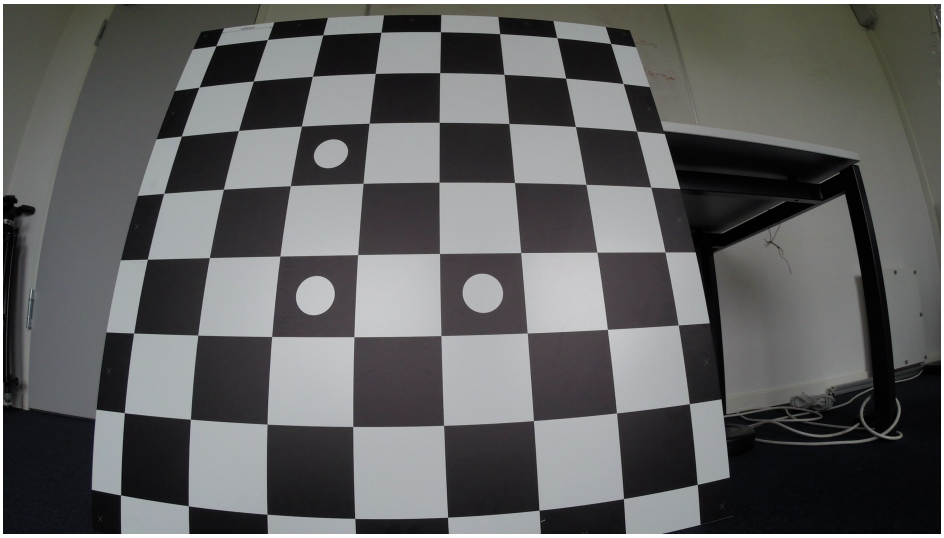


Abbildung 7.6 Ein Beispielbild aus den Bilddatensätzen, die für die intrinsische Kamerakalibrierung verwendet wurden.

Über ein solches Muster lässt sich über mehrere Aufnahmen eine Abdeckung des ganzen Bildes erreichen, was eine qualitativ hochwertige Kalibrierung zur Folge hat. Die resultierenden Parameter sind:

$$f_x = 1276.30580940725 \qquad k_4 = 0.0882443839204077$$
$$f_y = 1280.488858563099 \qquad k_5 = -0.1159228529802581$$
$$c_x = 1352 \qquad k_6 = 0.01672803249513818$$

$$
\begin{aligned}
c_y &= 762 & s_1 &= 0.009687755794654796 \\
k_1 &= -0.1754730639506888 & s_2 &= -0.00121437383815567 \\
k_2 &= -0.02682751034111265 & s_3 &= -0.0004370283270399302 \\
p_1 &= 5.92088436208239 \cdot 10^{-05} & s_4 &= 0.0002151082883221334 \\
p_2 &= -0.001371317353904053 & \tau_x &= 0.01337725664400075 \\
k_3 &= 0.007376263427206654 & \tau_y &= 0.0138165850489331
\end{aligned}
$$

Mit diesen Parametern lässt sich nun ein Bild der Auflösung 2704×1524, wie in Abb. 7.7 dargestellt, entzerren. Das Ergebnis ist qualitativ deutlich besser als das in Abb. 7.5 und ist für die nachfolgende Kalibrierung wesentlich komfortabler.



(a) Verzerrt (b) Entzerrt (c) Zugeschnitten

Abbildung 7.7 Die Entzerrungsphasen eines Bildes der Auflösung 2704×1524.

Die erhaltenen Parameter lassen sich mit einer skalierten Kameramatrix auch auf die Auflösung 1920×1080 anwenden und liefern, wie in Abb. 7.8 zu sehen, auch dort gute Ergebnisse.



Abbildung 7.8 Das Ergebnis der Entzerrung eines Bildes der Auflösung 1920×1080.

7.3 Extrinsische Kamerakalibrierung

Während der erste Datensatz der intrinsischen Kalibrierung zur Generierung von unverzerrten Bildern und Berechnung einer grundlegenden Kameramatrix dient, wird noch ein zweiter Datensatz benötigt, um Projektionen im verwendeten Bild zu ermöglichen. Diese Projektionen haben in dieser Arbeit eine wesentliche Bedeutung, da sie erst die Generierung von Referenzdaten ermöglichen und die hier berechneten Parameter für deren Präzision ausschlaggebend sind. Für eine korrekte Projektion im Sinne der Lochkamera werden sowohl intrinsische als auch extrinsische Parameter benötigt. Daher wird mit diesem System auch eine manuelle Kalibrierung auf dem entzerrten Bild ermöglicht. Das Muster hat sich jedoch verändert, da man nicht unbedingt viele Linienpunkte abdecken möchte, sondern möglichst genaue Referenzpunkte für die Kalibrierung haben möchte. Im Laufe der Entwicklung wurde das Muster in Abb. 7.9 analog zu der intrinsischen Kalibrierung mit *calibrateCamera* benutzt.

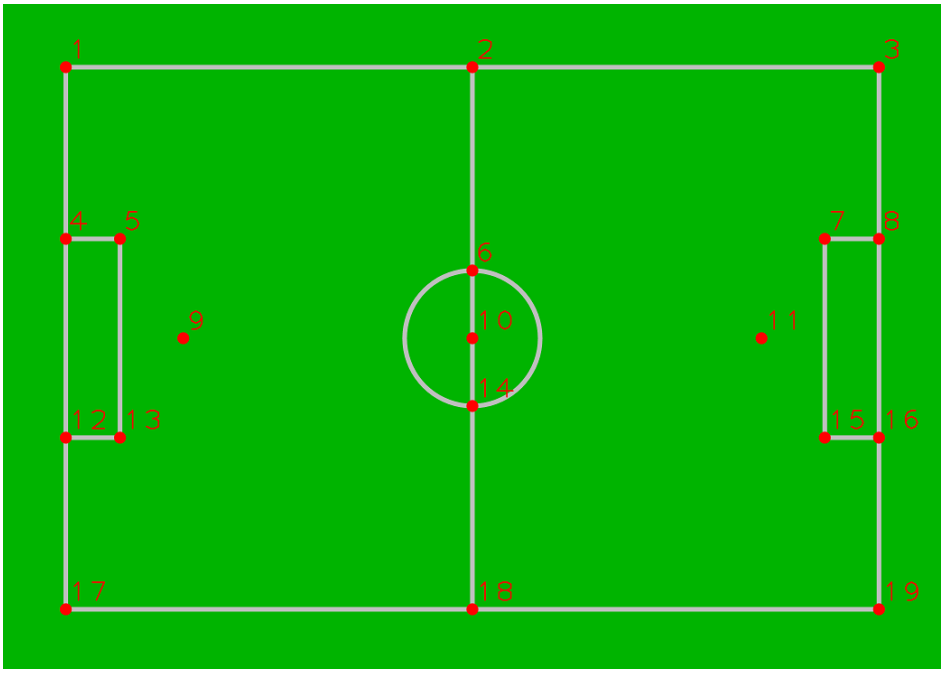


Abbildung 7.9 Das allgemeine extrinsische Kalibrierungsmuster.

Wenn man die *OpenCV* Dokumentation zur Kalibrierung [OpenCV, 2017a] liest, liegt es eigentlich näher, die intrinsischen Werte aus *getOptimalNewCameraMatrix* zu verwenden und wie empfohlen die extrinsische Ergänzung mit *solvePnP* zu kalibrieren, welches sich als nicht praktikabel erwies, da, wie sich bei späteren Recherchen herausstellte, der *Perspective-n-Point*-Algorithmus voraussetzt, dass die Eingabepunkte nicht in der selben Ebene liegen. Da dies bei Punkten auf einem Spielfeld nicht gegeben ist, lässt sich dieses Verfahren nicht anwenden und man muss die vollständige Kalibrierungsroutine mit *calibrateCamera* benutzen, welche dann die bisherigen intrinsischen Werte für das entzerrte und auf das Feld fokussierte Bild verfeinert. Bei späteren Testläufen wurde mit einer verringerten Anzahl an Kalibrierungspunkten getestet, wodurch sich das Muster in Abb. 7.10, auf Bildern mit einer

ähnlichen Perspektive wie in Abb. 7.8, als ebenso anwendbar wie das Muster in Abb. 7.9 herausstellte. Dieses Muster verringert den Zeitaufwand bei der Kalibrierung beträchtlich, da sich alle Punkte, wenn keine Verdeckungen vorhanden sind, präzise markieren lassen und die häufig aus perspektivischen Gründen fehlenden Eckpunkte nicht enthalten sind, weshalb man der Software keinen illegalen Punkt, der automatisch aussortiert wird, unterschieben muss. Dennoch sollte man bei einer deutlich abweichenden Perspektive das Muster aus Abb. 7.9 verwenden, da durch die Vielzahl der Punkte die mangelnde Präzision bei der Selektion weniger Punkte durch die präzise Selektion der Anderen ausgeglichen werden kann.

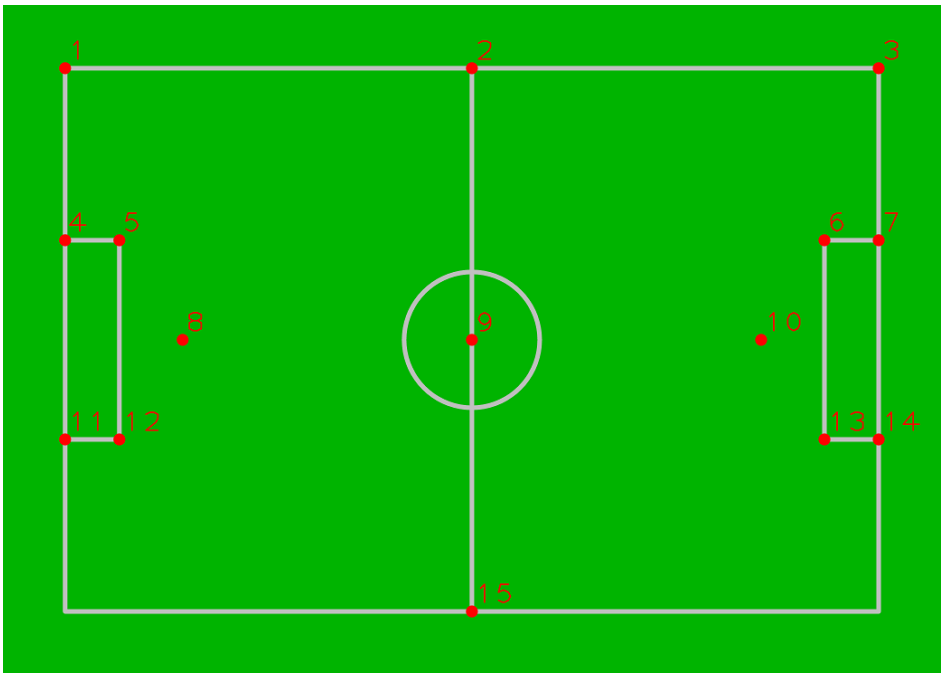


Abbildung 7.10 Das minimierte extrinsische Kalibrierungsmuster.

Um zu verdeutlichen, wie wichtig eine präzise Projektion ist, kann man die Welt-Zu-Kamera-Projektion der Feldlinien in einem frühen Entwicklungsstadium heranziehen. In Abb. 7.11 wurde auf einem nur näherungsweise entzerrtem Bild kalibriert, wobei für diese Kalibrierung auch wenige Verzerrungsparameter erlaubt waren. Bei der Betrachtung der Projektion stellt man fest, dass die Projektionen der nicht sichtbaren Eckpunkte außerhalb des Bildes liegen. Dieser Effekt ist nicht wünschenswert und liegt wahrscheinlich darin begründet, dass die Eckpunkte nicht in der Kalibrierung enthalten sind und durch die Möglichkeit der Verzerrung eine dreidimensionale Raumkrümmung ermittelt wurde. Diese Kalibrierung würde zur Folge haben, dass Roboterpositionen (in Weltkoordinaten), die nahe der unteren Ecke sein sollten, durch eine Projektion in das Bild weit im Feld sind.
Eine deutlich bessere Kalibrierung ist in Abb. 7.12 zu sehen, welche auf einem deutlich entzerrten Bild und der Annahme, dass keine Verzerrung mehr vorhanden ist, kalibriert wurde. Der eingezeichnete Fehler in Abb. 7.11 und Abb. 7.12 ist der von *OpenCV* berechnete *Root-Mean-Squared* (RMS) -Fehler, bei dem wie in Formel 7.1 beschrieben, der Unterschied zwischen dem projizierten Wert x_i mit der gewünschten Projektion $\hat{x}_i$ über alle Vorkommen

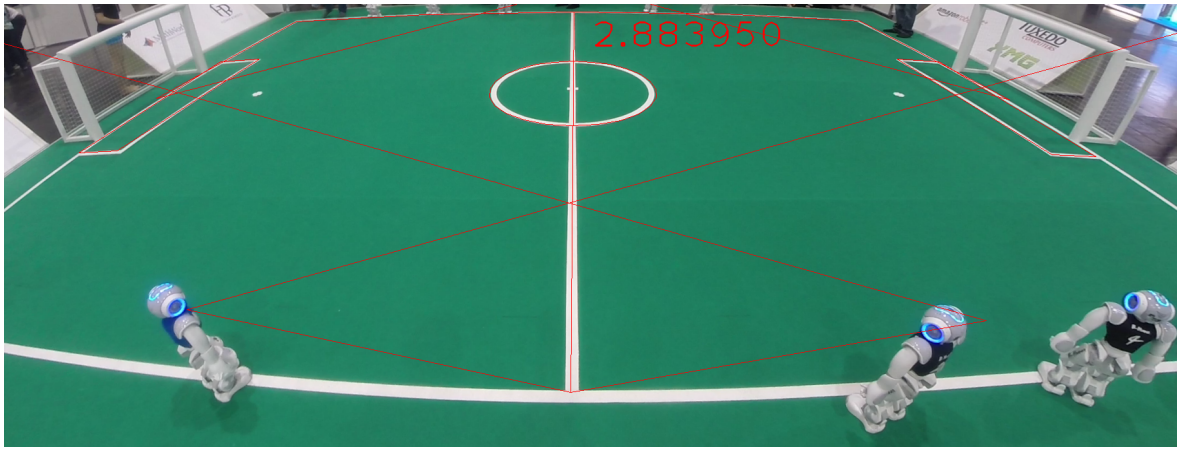


Abbildung 7.11 Der Projektionstest einer unvorteilhaften Kalibrierung, bei dem die erwarteten Feldlinien mit dem von *OpenCV* ermittelten Projektionsfehler in das Bild gezeichnet wird.

berechnet wird.

$$RMS_{Error} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{x}_i - x_i)^2} \tag{7.1}$$

Der RMS-Fehler gibt an, wie weit die Projektion durchschnittlich von ihrem Zielpunkt (nach Kalibrierung) entfernt ist. Ein RMS-Fehler von 5 gibt damit an, dass die durchschnittliche Distanz der Projektion von der ursprünglichen Kalibrierungseingabe 5 Pixel entfernt ist. Dass der Fehler in Abb. 7.11 kleiner als der in Abb. 7.12 ist, ist dem Umstand geschuldet, dass die erste Kalibrierung mit dem kleineren extrinsischen Muster kalibriert wurde, wodurch weniger Punkte miteinander verglichen werden und keine Referenzpunkte für die Extremfälle in den unteren Ecken gegeben sind.

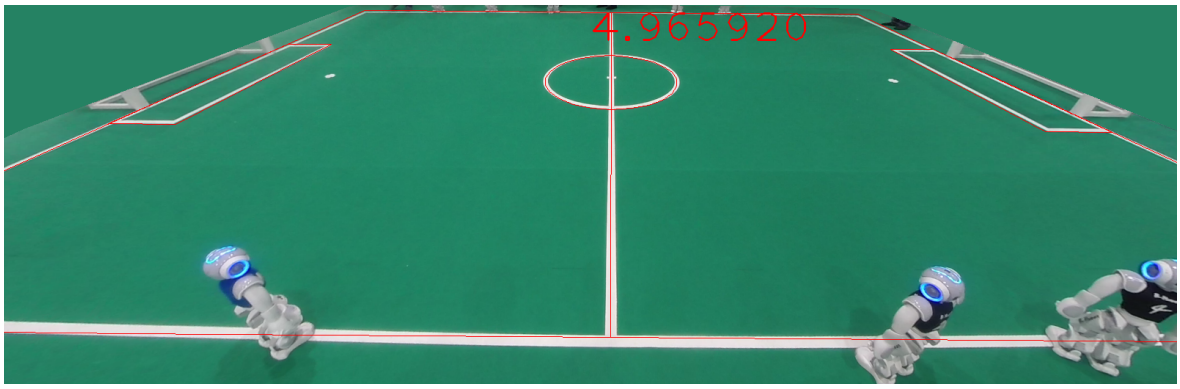


Abbildung 7.12 Der Projektionstest einer anwendbaren Kalibrierung.

7.4 Transformationen zwischen den Koordinatensystemen

Mit den gegebenen Kalibrierungen aus den vorherigen Abschnitten sind alle Parameter gegeben, um die beiden grundlegenden Transformationen in dem System zu beschreiben, die das Generieren der Referenzdaten ermöglichen.

7.4.1 Welt-Zu-Kamera

Diese Transformation wird dazu benutzt, eine Schätzung in Form einer Weltkoordinate in das Bild zu projizieren. Eine Anwendung dieser Projektion ist in Abb. 7.12 zu sehen, bei der die Feldlinien in das Bild projiziert wurden. Diese Transformation wird von *OpenCV* bereitgestellt und in OpenCV [2017a] beschrieben. Die Transformation kann mathematisch mit allen Verzerrungsparametern wie folgt beschrieben werden, wenn $z \neq 0$, also der zu projizierende Punkt vor der Kamera liegt:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = R \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} + t \tag{7.2}$$

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} \cdot \frac{1}{z} \tag{7.3}$$

$$r^2 = x'^2 + y'^2 \tag{7.4}$$

$$\begin{pmatrix} x'' \\ y'' \end{pmatrix} = \begin{pmatrix} x' \cdot \frac{1+k_1r^2+k_2r^4+k_3r^6}{1+k_4r^2+k_5r^4+k_6r^6} + 2p_1x'y' + p_2(r^2+2x'^2) + s_1r^2 + s_2r^4 \\ y' \cdot \frac{1+k_1r^2+k_2r^4+k_3r^6}{1+k_4r^2+k_5r^4+k_6r^6} + p_1(r^2+2y'^2) + 2p_2x'y' + s_3r^2 + s_4r^4 \end{pmatrix} \tag{7.5}$$

$$R(\tau_x, \tau_y) = \begin{pmatrix} \cos(\tau_y) & 0 & -\sin(\tau_y) \\ 0 & 1 & 0 \\ \sin(\tau_y) & 0 & \cos(\tau_y) \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\tau_x) & \sin(\tau_x) \\ 0 & -\sin(\tau_x) & \cos(\tau_x) \end{pmatrix} \tag{7.6}$$

$$= \begin{pmatrix} \cos(\tau_y) & \sin(\tau_y)\sin(\tau_x) & -\sin(\tau_y)\cos(\tau_x) \\ 0 & \cos(\tau_x) & \sin(\tau_x) \\ \sin(\tau_y) & -\cos(\tau_y)\sin(\tau_x) & \cos(\tau_y)\cos(\tau_x) \end{pmatrix} \tag{7.7}$$

$$s \begin{pmatrix} x''' \\ y''' \\ 1 \end{pmatrix} = \begin{pmatrix} R_{33}(\tau_x, \tau_y) & 0 & -R_{13}(\tau_x, \tau_y) \\ 0 & R_{33}(\tau_x, \tau_y) & -R_{23}(\tau_x, \tau_y) \\ 0 & 0 & 1 \end{pmatrix} R(\tau_x, \tau_y) \begin{pmatrix} x'' \\ y'' \\ 1 \end{pmatrix} \tag{7.8}$$

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \end{pmatrix} \cdot \begin{pmatrix} x''' \\ y''' \\ 1 \end{pmatrix} \tag{7.9}$$

Diese Gleichungen ermöglichen die vollständige Projektion mit der Nutzung aller Verzerrungsparameter und einem Skalierungsparameter s. Dabei werden zwölf der Parameter in Formel 7.5 benutzt, während die letzten zwei in Formel 7.6 und folgenden benutzt werden. Zu beachten ist, dass durch Formel 7.5 die Transformation nichtlinear wird. Die Verzerrungen

haben allerdings bei den Projektionen deutlich weniger Bedeutung, da die Kalibrierung, die für die Projektionen ausschlaggebend ist, auf dem „unverzerrten“ Bild ausgeführt wurde. Die Projektion ohne Verzerrungen sieht dadurch wie folgt aus:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = R \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} + t \tag{7.10}$$

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} \cdot \frac{1}{z} \tag{7.11}$$

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \end{pmatrix} \cdot \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} \tag{7.12}$$

7.4.2 Kamera-Zu-Welt

Die Gegenrichtung zu der vorherigen Projektion dient zur Generierung der Referenzdaten, da man ein Objekt im Bild erkennt und dessen Position in Weltkoordinaten bestimmen möchte. Diese Transformation ist nicht in *OpenCV* enthalten und wurde vom Benutzer Tetragramm auf GitHub [2017] vorgestellt. Die dort vorgestellte Projektion wurde adaptiert, um den Anforderungen dieses Systems zu genügen und lässt sich in mathematischer Form wie folgt beschreiben:

$$\begin{pmatrix} x'' \\ y'' \\ z'' \end{pmatrix} = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix}^{-1} \cdot \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \tag{7.13}$$

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = R^T \cdot \begin{pmatrix} x'' \\ y'' \\ z'' \end{pmatrix} \tag{7.14}$$

$$T_C = -R^T \cdot t = \begin{pmatrix} x_c \\ y_c \\ z_c \end{pmatrix} \tag{7.15}$$

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \frac{-z_c}{z'} \cdot \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} + \begin{pmatrix} x_c \\ y_c \\ z_c \end{pmatrix} \tag{7.16}$$

Auf den Bildpunkt wird die Gegenfunktion der Kameramatrix angewandt, um den Punkt von Bild- in Kamerakoordinaten zu transformieren. Das Resultat wird dann über die extrinsischen Parameter und die berechnete Kameratranslation in das Weltkoordinatensystem projiziert und dann skaliert, so dass der resultierende Punkt $z = 0$ besitzt, da er auf einer Ebene liegen soll.

7.5 Diskussion

Zur allgemeinen Kalibrierung gibt es viel Literatur. Allein Hartley und Zisserman [2011] präsentierten viele Algorithmen, um Kalibrierungen zu ermöglichen, aber durch die Verwendung von *OpenCV* liegt die Nutzung vorhandener Kalibrierungsalgorithmen nahe. Dennoch wäre es in weiteren Entwicklungen naheliegend, die Kalibrierung in der Form zu automatisieren, dass in dem ersten Bild die wichtigsten Punkte erkannt werden und für die Kalibrierung markiert und damit dem Kalibrierungsverfahren übergeben werden. Damit wäre diese Kalibrierung genauso automatisiert wie die Farbkalibrierung und würde damit den Aufwand des Nutzers deutlich reduzieren. Im Rahmen der Entwicklung wurde ein automatischer Ansatz zu einem frühen Entwicklungsstand auf einem verzerrten Bild getestet, welcher sich nicht als allgemein anwendbar erwies.

Kapitel 8

Erfassung der Roboter

In diesem Kapitel werden die Bildverarbeitungsalgorithmen beschrieben, die die Detektion von Robotern ermöglichen. Dabei werden zuerst die möglichen Ansätze betrachtet und dann die tatsächliche Umsetzung beschrieben. Anschließend werden diverse Probleme, die aus dem System hervorgehen, diskutiert.

8.1 Diskussion der Möglichkeiten

Viele Robotererkennungsalgorithmen setzen auf Farbsegmentierung. Arbeiten in diesem Bereich sind zum Beispiel in der Small Size League vorhanden, wie in dem Small-Size-Visionsystem von Zickler u. a. [2010]. Dabei ist zu beachten, dass die Erkennung nur dann gute Ergebnisse liefert, wenn der Roboter ein eindeutiges Farbmerkmal besitzt, welches in der Umgebung nicht vorkommt. Diese Ansätze sind in diesen Fällen effizient anwendbar, wenn eine effiziente Segmentierung gegeben ist. Dieser Ansatz ist bei *NAO*s nicht direkt praktikabel, da ihre Primärfarbe Weiß ist und die sekundäre Farbe Grau, welche in der Umgebung je nach Beleuchtung übermäßig stark vertreten sind. Diese Eigenschaft wird nur durch die Trikots geändert, die in früheren Versionen der SPL-Regeln auf die Farben Blau und Rot beschränkt waren. Dadurch sind Farben gegeben, welche sonst nicht auf dem Feld vorhanden sind. Durch die aktuellen Regeln [RoboCup Technical Committee, 2017] sind die Triktotfarben nicht mehr derart beschränkt.
Ein anderer Ansatz wäre eine Detektion über Formen. Formen dienen häufig als weitere Merkmale, um bisherige Ergebnisse zu verifizieren, wie bei Wilking und Röfer [2005]. Auch die Entwicklung von Kaufmann u. a. [2005] benutzt Formeigenschaften, die jedoch von Neuronalen Netzen evaluiert werden. Während der Entwicklung wurde ein auf Kanten- und Formerkennung basierender Ansatz ausprobiert. In der Annahme, dass der Roboter aus einer Menge von Kreisen besteht, kann man die *houghCirles*-Funktion von *OpenCV* auf Kantenbilder anwenden. Die Annahme ist insofern gerechtfertigt, dass der *NAO* als humanoider Roboter einen runden Kopf besitzt und an dessen Seiten zwei runde Lautsprecher angebracht

sind, die in den meisten Fällen zu sehen sind. In der tatsächlichen Anwendung war die durch *OpenCV* bereitgestellte Formerkennung selbst mit Parametrisierung nicht ausreichend, um die Roboter präzise zu erkennen und neigte dazu, einige False-Positives zu detektieren. Daher wurde dieser Ansatz verworfen.
Eine andere Möglichkeit wäre es, maschinelles Lernen zur Roboterdetektion zu nutzen. In vielen Arbeiten wird eine Form des maschinellen Lernens verwendet, wie zum Beispiel bei Kaufmann u. a. [2005]. Die besonders erfolgreichen Arbeiten zeichnen sich durch eine Erkennungsrate von 90% aus, wobei man diese nur mit einer großen und vielseitigen Trainingsmenge erreichen kann. Um eine derartige Trainingsmenge zu erstellen, wird viel Material und sehr viel Aufwand benötigt, weshalb es den Rahmen dieser Arbeit sprengen würde. Dennoch wäre ein solcher Ansatz für künftige Entwicklungen denkbar. Ebenso wäre es möglich, ein Modul zu entwickeln, welches möglichst automatisiert Trainingsdaten ausgibt, welche dann gelabelt werden müssen.
Der in dieser Arbeit verfolgte Ansatz zur Detektion von Robotern basiert auf Kantendetektion und Farbsegmentierung, deren Robustheit durch die Farbkalibrierung in Kapitel 6 gegeben ist. Zusätzlich dazu wird Wissen über den Aufbau der Roboter mit perspektivischen Annahmen kombiniert, wodurch eine bessere Detektion von Robotern ermöglicht wird. Ergänzend dazu erwies sich das Aufsplitten der Detektion in die initiale Detektion und weiteres Tracking als äußerst hilfreich.

8.2 Initiale Erfassung der Roboter

Die initiale Erfassung von Robotern soll absolut sichere Perzepte von Robotern liefern. Um dies zu ermöglichen, wurde in Kapitel 6 das Bild auf einen relevanten Ausschnitt zugeschnitten und der immer noch sichtbare Hintergrundanteil übermalt, weshalb Phänomene wie Hintergrundstörungen ausgeschlossen sein sollten. Daher ist es naheliegend, auf einem Kantenbild nach Features zu suchen, welche zusammen einen Kandidaten ergeben, den man validieren möchte. In dieser Anwendung wird zur Berechnung von Kantenbildern der Sobel-Operator (vgl. Formel 4.33) verwendet, wodurch Bilder wie in Abb. 8.1 entstehen.

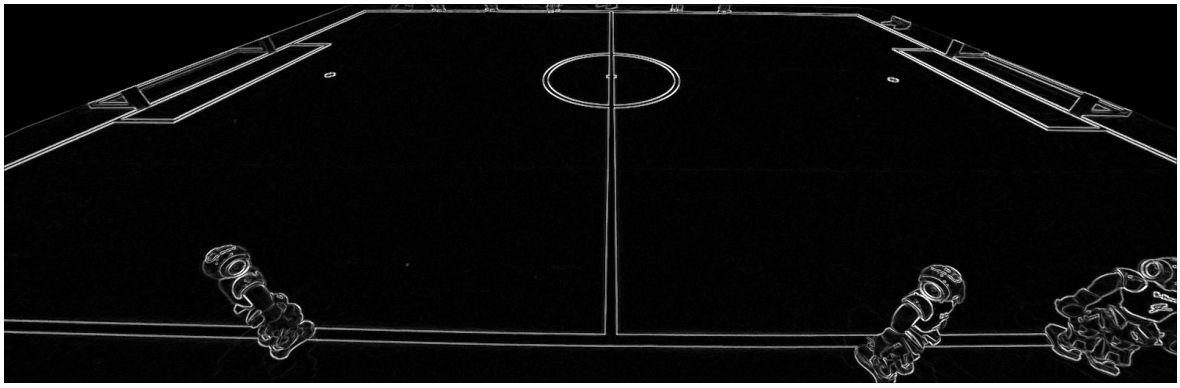


Abbildung 8.1 Das mit dem Sobel-Operator berechnete Kantenbild einer Szene.

Auf diesem Bild wird der FAST Algorithmus (vgl. Kapitel 4.7) als Feature-Detektor angewendet. Dadurch erhält man eine Liste von Merkmalen mit ihren Bildpositionen, wie in Abb. 8.2 eingezeichnet. Wie zu erwarten war, wurden sehr viele Feature detektiert. Die Roboter setzen sich aus sehr vielen Features zusammen, während andere Detektionen vereinzelt vorkommen oder an Feldlinien und für die Detektion irrelevanten Stellen gehäuft sind.

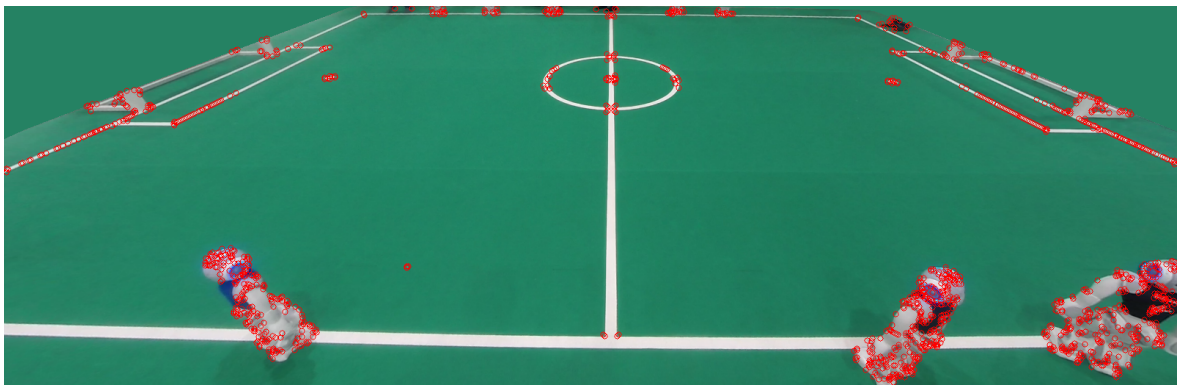


Abbildung 8.2 Die Resultate der FAST Anwendung in rot in das Originalbild eingezeichnet.

Aus diesen Features kann man nun Regionen bestimmen, die als Roboterkandidaten behandelt werden. Hierzu werden zwei Beobachtungen benutzt:

1. Die Features eines Roboters können nicht weit auseinander liegen. Hier ist die Distanz zwischen zwei Features, die zum selben Roboter gehören, kleiner als 20 Pixel.

2. Ein Roboter, selbst ein Ausschnitt, der nur die Füße betrachtet, besteht aus mehr als 8 Features.

Um die erste Beobachtung umzusetzen, wird die Struktur *FeatureDist* in Kombination mit der *OpenCV* Funktion *partition* angewendet. Dabei entstehen Cluster von Features, wie in Abb. 8.3 dargestellt, bei dem jedes Cluster eine zufällige Farbe erhalten hat.

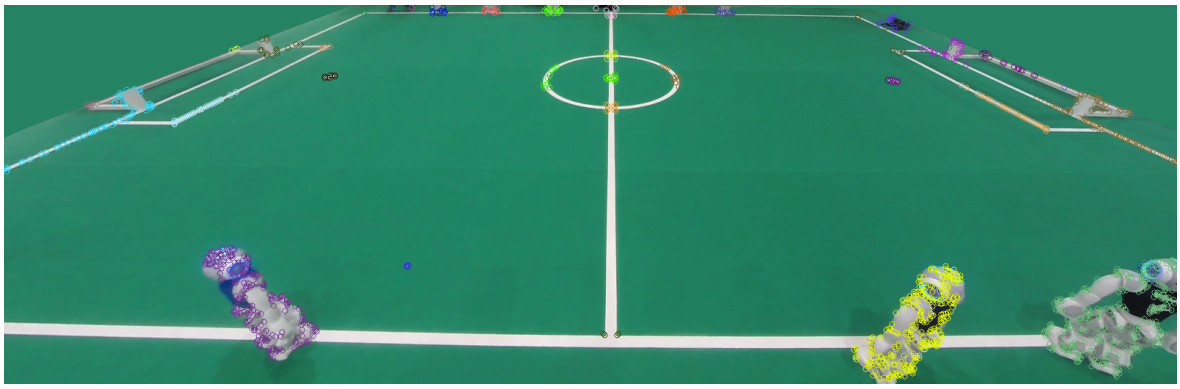


Abbildung 8.3 Beispiel des Resultats der *partition* Anwendung über einen Distanzschwellwert.

Wenn die Cluster dann auf eine Mindestanzahl von 8 Features überprüft werden und nur diese für die weitere Berechnung bleiben, sind nur noch Cluster in der Form wie in Abb. 8.4

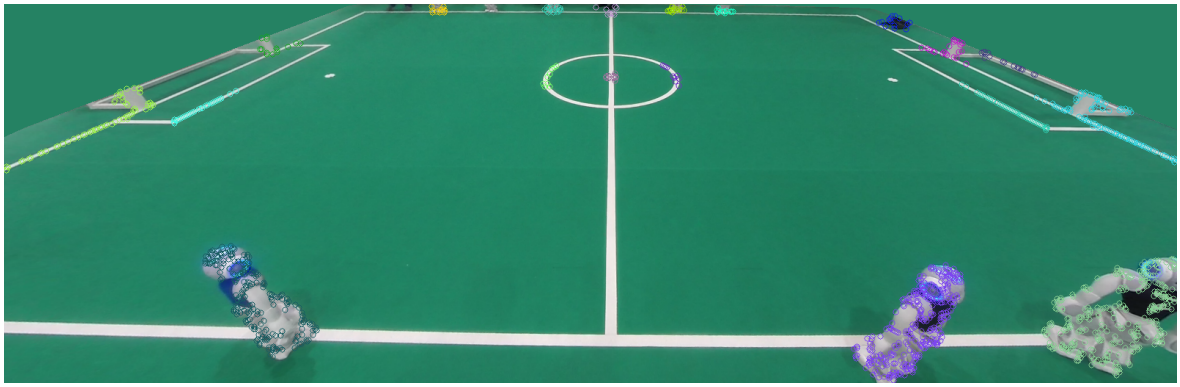


Abbildung 8.4 Alle Cluster aus Abb. 8.3, die eine Mindestanzahl von Features besitzen.

enthalten.
Mit diesem Clustering ist die Abtastung des vollständigen Bildes abgeschlossen. Danach werden die ganzen Cluster als Roboter-Kandidaten betrachtet und auf diverse Eigenschaften überprüft. Jedes Cluster repräsentiert eine Bildregion, also wird wie in Kapitel 6 wieder mit Region of Interest (ROI) gearbeitet. Diese ROIs lassen sich über *boundingRect* aller Punkte des Clusters bestimmen. Einige der generierten Kandidaten sind viel zu klein, um eine Aussage über den Inhalt zu treffen. Daher werden nur die ROIs weiter untersucht, deren Fläche größer gleich 1000 Pixel^2 sind. Diese Maßnahme wird mit der Funktion *verifyROI* realisiert. Die daraus generierten Kandidaten sind in Abb. 8.5 gezeigt.

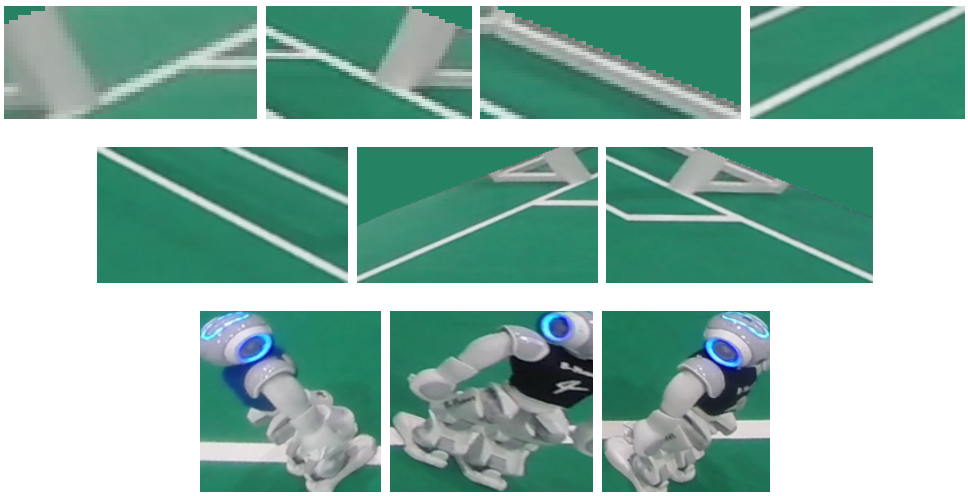


Abbildung 8.5 Die aus dem realen Bild extrahierten Roboter-Kandidaten.

Zu jedem dieser Kandidaten werden durch die Klasse *ImageGMM* zwei Binärbilder berechnet. Eines, welches den Schwarzanteil darstellt und eines, welches alle nicht grünen Pixel darstellt, also ein invertiertes Grün-Bild. Zu beachten ist, dass diese Bilder durch ein dreifaches Erodieren und ein einzelnes Dilatieren von Störungen im Bild befreit werden.
Um einen Fall der Verdeckung zu behandeln, wird jeder Kandidat mit *checkForSplit* auf einen möglichen benachbarten Roboter geprüft. Dies wird gemacht, in dem das Nicht-Grün-Bild

nach Konturen durchsucht wird. Diese Konturen werden dann nach Flächengröße sortiert. Wenn die größten Konturen in der Liste von der Fläche her zu nahe beieinander liegen, werden aus diesen nahezu gleichgroßen Konturen neue ROIs berechnet, die wieder der Kandidatenliste hinzugefügt werden. Anschließend wird der aktuelle Kandidat verworfen.
In der Konturüberprüfung *contourExtraction* wird auch die größte Kontur im Schwarz-Bild berechnet. Über einige Annahmen über den Bildaufbau der Kandidaten-Region lassen sich einige Kriterien für valide Roboter formulieren.

1. Die Kontur im Nicht-Grün-Bild muss existieren: Das Kriterium stellt sicher, dass eine Kontur auf dem Grün existiert, welche als Roboter in Frage kommt.

2. Die Kontur im Schwarz-Bild muss existieren: Dieses Kriterium besagt, dass man davon ausgeht, bei einem validen Roboterkandidaten ein Stück des Trikots im Bild zu haben.

3. Es müssen grüne Pixel in der Region existieren: Wenn man die Regionen in Abb. 8.5 betrachtet, sind alle Roboter von Grün umgeben. Wenn man versucht, ein nicht rotiertes umschließendes Rechteck für die Features der Roboter zu finden, ist es nahezu unmöglich, jegliches Grün auszuschließen.

4. Der Grünanteil im ROI sollte nicht 95% übersteigen: Über einen so kleinen Bereich im Bild lässt sich, analog zu kleinen ROIs, kaum eine Aussage treffen.

5. Der Schwarz-Anteil der Nicht-Grün-Fläche sollte nicht größer als $\frac{1}{3}$ sein: Vorausgesetzt, dass Schwarz das Trikot repräsentiert, ist dies eine sinnvolle Annahme. Der Schwellwert war ursprünglich toleranter gegenüber größeren Schwellwerten, wurde aber in der Entwicklung so optimiert, dass dadurch einige mögliche False-Positives gefiltert werden.

6. Die Fläche der Nicht-Grün-Kontur muss größer sein als die der Schwarz-Kontur: Dies folgt aus den vorherigen Annahmen, wurde jedoch benutzt, als andere Formen der Farbklassifizierung verwendet wurden. Zusätzlich stellt es sicher, dass beide Konturen zu demselben Objekt in der Region gehören.

7. Die Anzahl der schwarzen Pixel im Bild darf nicht größer sein als 25% der Nicht-Grün-Pixel im Bild: Ein Kriterium, um zu vermeiden, dass ein Schatten unter den Roboterfüßen für ein Robotertrikot gehalten wird und um Schiedsrichter auszuschließen.

Die Kandidaten, die alle diese Kriterien erfüllen, wurden in Abb. 8.6 für Debugging-Zwecke festgehalten und spiegeln die ungefähre Erwartungshaltung über valide Robotern wieder. Die größte Schwarz-Kontur ist als blaue Linie und die größte Nicht-Grün-Kontur als rote Linie eingezeichnet. Zur Untersuchung der Größenverhältnisse wird in die Mitte des Bildes der Wert $\frac{\text{Nicht-Grün-Fläche}}{\text{Schwarz-Fläche}}$ und $\frac{\text{Nicht-Grüne-Pixel}}{\text{Schwarze-Pixel}}$ (von oben nach unten) geschrieben, während oben links die Fläche der größten Schwarzkontur und oben rechts die Fläche der größten Nicht-Grün-Kontur steht. Ergänzend dazu steht unten links die Anzahl der erlaubten grünen Pixel und auf der rechten Seite die der gefundenen grünen Pixel. Die Kandidatenanzahl wurde von

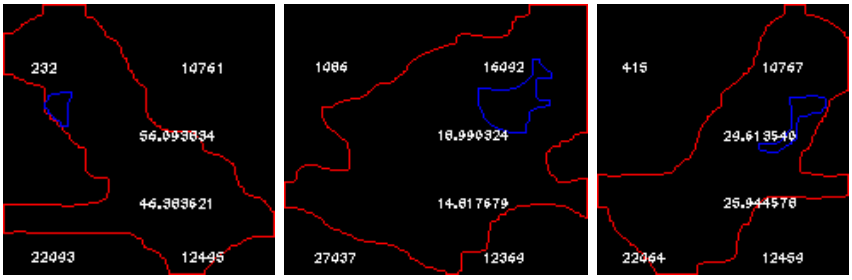


Abbildung 8.6 Die Kontur-Debug-Bilder der validen Roboter-Kandidaten.

10 in Abb. 8.5 auf 3 in Abb. 8.6 reduziert.
In der nächsten Überprüfung wird der Fußpunkt des Roboters bestimmt. Für die Fußpunktdetektion wird die Nicht-Grün-Kontur des Kandidaten in ein separates Bild gezeichnet. Die Funktion *extractFeetCenter* versucht dafür, eine repräsentative Form in diesem Bild zu finden. Eine Funktion, die eine solche Form annähern kann, ist *fitEllipseDirect*. Dieser Funktion werden alle Pixelpositionen der weißen Pixel im Nicht-Grün-Kontur-Bild übergeben. Die Ellipse wird dann in Form eines rotierten Rechtecks, welches die Ellipse umschließt, zurückgegeben. Ein solches Rechteck ist vor allem eine gute Repräsentation, wenn man davon ausgeht, dass der Roboter gänzlich im Bild ist. Auf diese Repräsentation des Roboters wird nun eine Überprüfung der Dimensionen ausgeführt, bei dem alle Rechtecke, deren größere Seite um mehr als den Faktor 10 länger ist als die kürzere Seite, verworfen werden und somit eine Detektion verhindern. Das Rechteck zum Repräsentieren des Roboters wird ansonsten für die ROI-Findung verwendet. Da perspektivisch die Annahme getroffen werden kann, dass sich der relevante Teil zur Fußpunktberechnung im unteren Abschnitt der Bildregion befindet, wird das unterste Viertel des Bildes extrahiert. Auf diesem Ausschnitt wird ein Mittelpunkt über alle weißen Pixel ermittelt. Falls sich keine weißen Pixel in der Region befinden, schlägt die Detektion fehl. Der dadurch berechnete Fußpunkt wird benutzt, um das repräsentative Rechteck zu erweitern. Einige Aspekte dieser Untersuchung sind in Abb. 8.7 abgebildet.

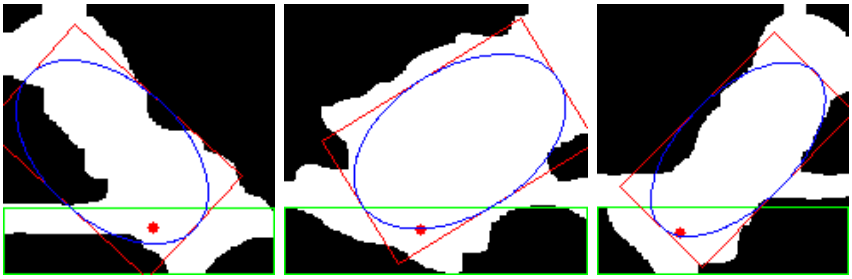


Abbildung 8.7 Die Positions-Debug-Bilder der validen Roboter-Kandidaten. Dabei ist das repräsentative Rechteck in Rot, dessen erzeugen Ellipse in Blau und der Fußausschnitt in Grün eingezeichnet. Der detektierte Fußpunkt ist als roter Punkt markiert.

Der resultierende Punkt durchläuft einen finalen Test, bei dem sichergestellt wird, dass der Bildpunkt innerhalb des Bildes liegt. Diese Überprüfung wird benutzt, da andere Ansätze

der Fußpunktdetektion an Spezialfällen am Rand des Bildes einen unmöglichen Fußpunkt zurückgeliefert haben. In der jetzigen Variante ist dieses Phänomen ausgeschlossen. Anschließend wird der Bildpunkt in Weltkoordinaten projiziert. Der dadurch berechnete Punkt muss für eine zulässige Detektion auf dem Feld-Teppich sein und innerhalb der Teppichgrenzen in Weltkoordinaten liegen. Dieser finale Test stellt sicher, dass unzulässige Detektion, die an den Rändern des Bildes zufällig alle anderen Kriterien erfüllen, aussortiert werden. Das Ergebnis eines solchen Detektionsdurchlaufs sind die Fußpositionen der Roboter und deren ROIs, die zuvor berechnet wurden. Die Liste der Fußpunkte wird direkt an den *HypothesesManager* weiter geleitet und analysiert, während die Liste der ROIs im *RobotDetector* für das Tracking benutzt wird. Diese Resultate sind in Abb. 8.8 eingezeichnet.



Abbildung 8.8 Die Ergebnisse der Roboterdetektion. In Blau sind die ROIs der detektierten Roboter und in Rot die Fußpunkte gezeichnet.

Wenn man die Resultate in Abb. 8.8 betrachtet, so stellt man fest, dass eine angemessene Fußpunktdetektion bei den vollständig zu sehenden Robotern möglich ist. Die Detektion der Roboter, von denen nur die Füße sichtbar sind, beinhaltet viele Grenzfälle, die zu False-Positives führen können. Eine Erweiterung des Sichtfeldes in der Form, dass alle Roboter zur Gänze zu sehen sind, ist in vielen Fällen nicht sinnvoll, da dadurch das Bild deutlich an Hintergrundstörungen gewinnt. Dieser initiale Detektionsansatz setzt damit deutlich auf vollständig sichtbare Roboter und trifft einige Annahmen über deren Anatomie sowie darüber, wo perspektivisch deren Fußpunkte liegen müssten. Ein Sonderfall sind liegende Roboter, deren Fußpunkt damit je nach Lage im Kopf, in den Füßen oder in der Körpermitte liegen kann, wenn die Detektion diese überhaupt erkennt. Dieser Sonderfall muss in der Detektionsevaluation in Kapitel 10.2 berücksichtigt werden.

8.3 Tracking bereits erfasster Roboter

Während der initiale Prozess für den Nachweis eines Roboters im Bild recht aufwändig ist, sollte es wesentlich einfacher sein, einen Roboter nachzuweisen, den man zuvor schon einmal detektiert hat. In diesem Sinne werden alle ROIs bereits detektierter Roboter gespeichert und für die erneute Überprüfung in *checkTracked* verwendet, wie in Abb. 8.9 (a) dargestellt.

Die Region wird verifiziert, indem ein von Störung befreites Nicht-Grün-Bild analog zu der initialen Detektion erstellt wird, wie in Abb. 8.9 (b) abgebildet. Auf diesem Bild wird die größte Kontur gesucht und anschließend die Funktion *extractFeetCenter* von der initialen Detektion benutzt, um den Fußpunkt zu bestimmen. Das Tracking soll auch die Bewegung des Roboters kompensieren, weshalb ein ROI nicht statisch sein darf. Deswegen wird das repräsentative Rechteck von *extractFeetCenter* benutzt, um den ROI zu aktualisieren, welches in Abb. 8.9 (c) rot eingezeichnet ist. Das Rechteck dient der Sicherheit, dass die Region nicht zu groß wird, und wird für die Berücksichtigung von Bewegungen mit heuristischen Werten skaliert und dann über *boundingRectangle* als ROI benutzt. Dieser ROI wird dann auf das Bild zurechtgeschnitten, falls es rechnerisch aus dem Bild herausragen würde. Wenn der resultierende ROI mehr als $\frac{1}{5}$ der Bildfläche einnimmt, wird der getrackte Roboter verworfen, ansonsten wird der alte ROI durch den neuen ersetzt. Zusätzlich wird überprüft, ob der errechnete Fußpunkt sich im Bild befindet. Dieses Verhalten bewirkte bei Videos, die bei 30 FPS aufgenommen wurden, ein sinnvolles Trackingverhalten.



Abbildung 8.9 Bildreihe, die für das Tracking eines Roboter erstellt wird. In (a) der Ausschnitt des echten Roboters, in dem ein Roboter verifiziert werden soll. (b) stellt das Nicht-Grün-Bild des Ausschnittes dar. (c) ist die Debug-Sicht der Fußpunktdetektion.

8.3.1 Resultierende Änderungen in der Detektion

Durch die Teilung der gesamten Robotererfassung in eine initiale Detektion und das weitere Tracking verändert sich auch nach einem Trackingdurchgang die initiale Detektion. Zum Beispiel ist es nicht wünschenswert, wenn in der anschließenden Detektion dieselbe Region nochmal untersucht wird und dafür wiederholt ein Roboter eingetragen wird. Um dies zu vermeiden, wird das Sobel-Bild vor der initialen Robotererfassung so modifiziert, dass die Regionen der bereits erfassten Roboter dort schwarz eingezeichnet werden, wie in Abb. 8.10 dargestellt.
Durch die anschließende FAST Anwendung ist es nahezu ausgeschlossen, dass ein Roboter so auf die umständliche Weise wiederholt erfasst wird. Dennoch kann es sein, dass durch eine ungünstige Konstellation der Features eine Region entsteht, die einen bereits erfassten

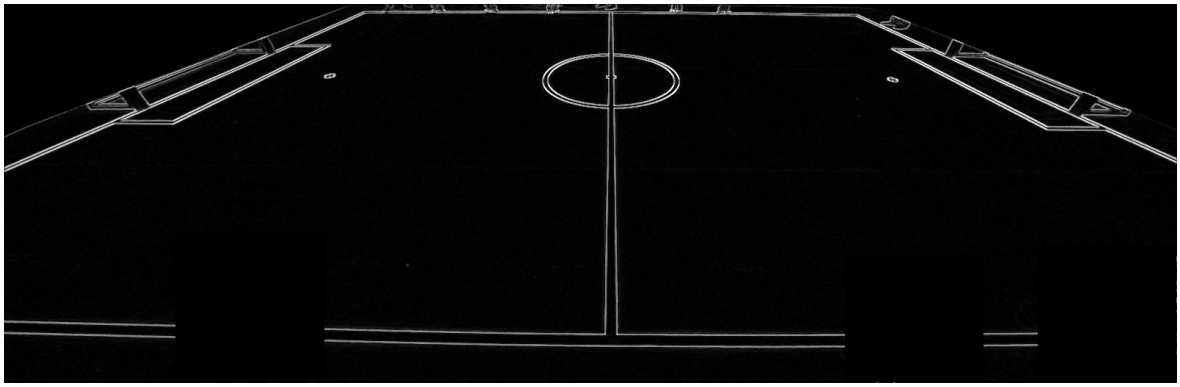


Abbildung 8.10 Das modifizierte Kantenbild nach der Durchführung des Trackings.

Roboter umschließt. Um damit nicht unnötig viel zu rechnen und mögliche False-Positives zu riskieren, wird bei jeder Region eines Kandidaten mit *checkIfAlreadyTracked* verifiziert, ob er einen bereits erfassten Roboter umschließt oder gar von diesem umschlossen wird, indem die Schnittmenge der Rechtecke überprüft wird. Dabei behandelt die Funktion durch die Vergrößerung des ROIs, um zwei Pixel in jede Richtung, auch einen Sonderfall, der durch eine zu kleine Tracking-ROI zustande kommt. Ein zu kleiner ROI für das Tracking kann darin resultieren, dass Unter- und Oberkörper des Roboters separat erfasst werden. Durch die initiale Vergrößerung des neu erfassten Kandidaten wird dieser dann verworfen, wenn er zu nahe am alten Kandidaten ist. Dennoch ist es möglich, dass durch diesen Fehlerfall sowohl Unter- als auch Oberkörper erfasst werden, wenn eine gewisse Distanz zwischen den beiden Regionen besteht.
Der gesamte Detektionsablauf wird in Abb. 8.11 visualisiert. Das Resultat der Routine ist eine Liste von Bildpunkten, die zur Generierung von Referenzdaten an den *Hypothesesmanager* weiter gegeben wird.

8.4 Betrachtung möglicher Probleme

Mit dem Design einer Detektionsroutine werden spätestens bei den ersten Testläufen Probleme ersichtlich. Ein größeres Problem in diesem System sind Verdeckungen, welche zumindest ein Perzept von den prinzipiell Möglichen nicht detektierbar machen und im schlechtesten Fall sogar noch ein anderes verfälschen. Das Verdeckungsproblem wird hier durch eine Kombination der Szenenzerlegung und der Konturfindung verursacht. Die Funktion *checkForSplit* behandelt dabei nur einen einfachen Fall der Verdeckung durch Szenenzerlegung des Feature-Clusterings, bei dem zwei nahe beieinander stehende Roboter in dieselbe Region gefasst wurden und durch ihre annähernd gleich großen Konturen differenziert werden können. Dies ist eine unzureichende Lösung, da dies ein sehr spezieller Fall ist und viele andere Szenarien entstehen können, wie zum Beispiel unterschiedlich große Konturen der beiden Roboter oder eine Verdeckung, bei der nur eine Kontur vorhanden ist. Allein bei diesen zwei Fällen ist schon eine Auflösung der Verdeckung durch das hier vorgestellte System nicht mehr mög-

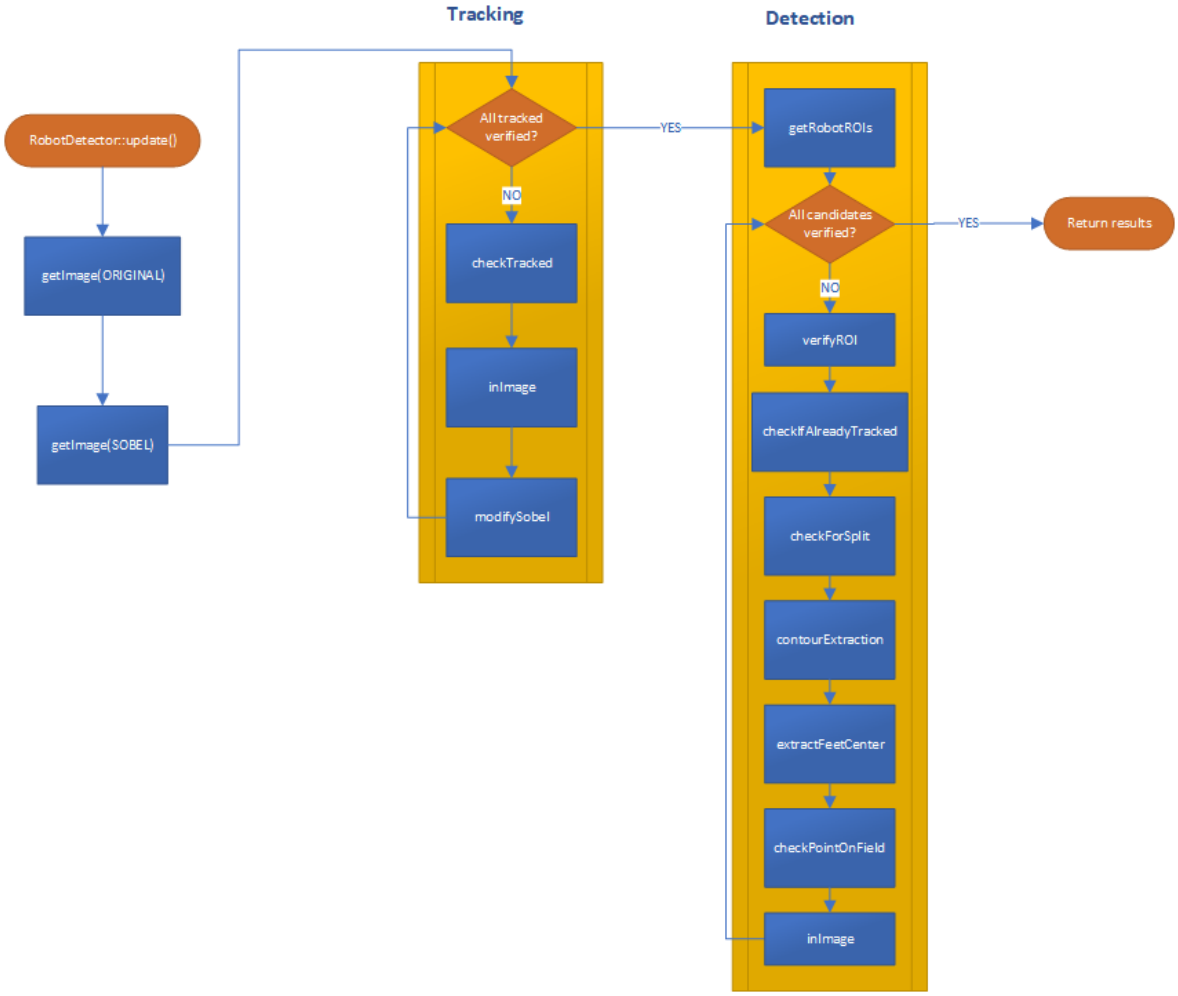


Abbildung 8.11 Der vollständige Detektionsablauf auf höchster Ebene visualisiert. Die blauen Prozeduren beschreiben wichtige Funktionsaufrufe, währen die orangenen Rauten die Entscheidungspunkte der Schleifen darstellen und die anderen Knoten den Anfangs- und Endpunkt. Eine fehlgeschlagene Funktion innerhalb einer Schleife bewirkt eine Rückkehr zum Entscheidungspunkt.

lich. Eine allgemeine Lösung für das Problem wäre die Verwendung von mehreren Kameras, welche das Spielgeschehen aus mehreren Perspektiven erfassen. Dadurch würden dem System viele Möglichkeiten gegeben, um etwaige Verdeckungsprobleme zu beheben. Eine andere Möglichkeit, um zumindest einen Teil der Probleme zu beheben, wäre ein Ansatz wie von Otake, Murakami und Naruse [2008] zu verwenden, bei dem Verdeckungen über statistische Merkmale der umgebenden Pixel erkannt werden. Für diese Arbeit gilt es, die Auswirkung der Verdeckung bei der Roboterperzeption zu eruieren, wie in Kapitel 10.7 dargestellt.
Eine besondere Herausforderung ist es, die Bilder zu identifizieren, in denen eine Detektion von Robotern nahezu unmöglich ist, da die Szene kaum ausgewertet werden kann. Diese Bilder können während bestimmter Spielphasen entstehen, in denen die Schiedsrichter über das Feld laufen und die Roboter korrekt anordnen, um einen ordnungsgemäßen Verlauf des Spiels zu garantieren. Dieses Problem würde sich durch eine Synchronisierung mit dem Nachrich-

tenverkehr des *GameControllers*, des Schiedsrichtertools im RoboCup, beheben lassen. Eine andere Situation, die die Detektion erschweren würde, wäre eine Verdeckung von außerhalb, wie zum Beispiel durch einen Zuschauer oder Schiedsrichter, die einen Großteil des Bildes einnehmen. Diese Fälle sind in Abb. 8.12 dargestellt. Wie stark diese Phänomene in der allgemeinen Auswertung der Spielszenen die Detektion erschweren, soll in Kapitel 10.3 zumindest partiell ersichtlich sein.



(a) (b)

Abbildung 8.12 Beispiele möglicher Verdeckungen, die nicht durch Roboter entstehen. Wobei (a) durch Synchronisation der Netzwerk-Nachrichten unterscheidbar wäre im Gegensatz zu (b).

Wie alle farbbasierten Detektionsansätze trifft auch dieses Verfahren eine Annahme über die zu detektierenden Farben, obwohl ein dynamisches Farbklassifizierungverfahren über ein *Gaussian Mixture Model* benutzt wurde. Die grundlegende Idee war es, durch die Schwarz-Farbklasse eine Trikoterkennung zu ermöglichen und darüber Roboter in den extrahierten Regionen zu identifizieren. Dieser Ansatz steht im Konflikt zu den aktuellen Regeln [RoboCup Technical Committee, 2017] und wäre für alte Regelversionen deutlich besser geeignet. In jüngerer RoboCup Geschichte wurde in der SPL die Benutzung von weißen und grünen Trikots erlaubt, welche in diesem System in eigenen Farbklassen gehandhabt werden und somit nicht als Trikotfarbe im Sinne von Schwarz behandelt werden können.
Ein weiterer Aspekt, der Unsicherheit in die Detektion von Robotern einbringt, ist die Anpassung des ROI im Tracking. Diese Problemstellung hat bei dem Detektionsalgorithmus nur einen engen Rahmen, da die Region groß genug sein soll, um den Roboter vollständig zu beinhalten und so klein, dass keine verfälschenden Informationen beinhaltet sind. Der Ansatz mit der Beschreibung des Körpers über eine Ellipse erwies sich in Testläufen als recht robust, aber führt bei Regionen, die einen größeren Abschnitt der Feldlinien beinhalten, zu Ellipsen, die durch den Roboter verlaufen und ansonsten dem Verlauf der Linien annähernd folgten. Dies führt zu Regionen, die im weiteren Videoverlauf immer größer werden und den Fußpunkt verfälschen. Eine andere Variante, die durch Erosion auf kleinen Bildern entsteht, wurde in Kapitel 8.3.1 beschrieben, bei der sowohl Ober- als auch Unterkörper eines Roboters erfasst werden, was durch einen zu kleinen ROI ermöglicht wurde. Wie gut diese Phänomene durch perspektivische Annahmen verhindert oder begünstigt werden, soll in Kapitel 10.6 untersucht werden.

Der hier vorgestellte Algorithmus hat durch seine Konstruktion eine gewisse Bandbreite von Vor- und Nachteilen, seine Anwendbarkeit muss allerdings mit weiteren Komponenten des Systems in Kapitel 10 evaluiert werden.

Kapitel 9

Modellierung der Roboter

In diesem Kapitel wird erläutert, wie aus den Bildpunkten die Referenzdaten generiert werden und mit welchen Maßnahmen sichergestellt wird, dass diese nicht entarten.

9.1 Der Aufbau des UKF

Für jeden erfassten Roboter soll ein *Unscented Kalman-Filter* (vgl. Kapitel 4.5.1) existieren, welches die Schätzung seines aktuellen Zustands repräsentiert. Dieser Zustand ist in diesem System ein Vektor

$$\mu_t = \begin{pmatrix} x \\ y \end{pmatrix} \tag{9.1}$$

, der die Position des Roboters in Weltkoordinaten (vgl. Kapitel 4.2.1) angibt. Dessen Unsicherheit Σ_t beträgt initial

$$\Sigma_t = \begin{pmatrix} 2500 & 0 \\ 0 & 2500 \end{pmatrix} \tag{9.2}$$

, woraus ersichtlich ist, dass die initiale Projektion als nicht besonders zuverlässig gilt. Dabei nimmt das *Unscented Kalman-Filter* als Messung einen Vektor

$$z_t = \begin{pmatrix} u \\ v \end{pmatrix} \tag{9.3}$$

entgegen, der das Zentrum der Fußpositionen des Roboters in Bildkoordinaten angibt. Durch diesen Aufbau sind Zustands- und Messraum sehr unterschiedlich, weshalb je nach Anwendungsform der Projektionen (vgl. Kapitel 7.4) eine Nichtlinearität gegeben ist. Deshalb bietet sich die Anwendung eines *Unscented Kalman-Filters* an, der diese Art von Funktionen in Aufrufen von g und h kapselt.

Der Dynamikschritt in dieser Systembeschreibung ist ein Sonderfall, da einerseits keine Dynamik direkt durch den Zustand und andererseits keine Zustandsübergangsmessung u_t gegeben ist. Dadurch ist die Zustandsübergangsfunktion g eine Identitätsfunktion. Jedoch kann es sein, dass von einem Ausführungsschritt zum nächsten der Roboter eine Bewegung gemacht hat, welche seine Schätzung verändern dürfte, weshalb dadurch die Unsicherheit in Σ_t steigt. Um dieses Wissen zu quantifizieren, wird R_t benutzt, welches die Unsicherheit in der Dynamik codiert. In diesem System erwies sich

$$R_t = \begin{pmatrix} 2000 & 1500 \\ 1500 & 2000 \end{pmatrix} \tag{9.4}$$

als anwendbar. Zusammengefasst bewirkt der Dynamikschritt keine Veränderung im Sinne der Schätzung μ_t, sondern lässt die Unsicherheit im Sinne von Σ_t steigen, was durch die Messung kompensiert werden soll.
Der Messschritt ist damit das Kernstück des Filters, der in der Klasse *UKF* umgesetzt ist. Die Sigmapunkte werden mit h über Formel 7.9 von Welt- zu Bildkoordinaten transformiert. Um die Unsicherheit in der Präzision der Roboterdetektion zu codieren, wird Q_t benutzt. Ein Problem, was aus der Transformation von Bildkoordinaten zu Weltkoordinaten erwächst, ist ein größerer Fehler bei steigender Entfernung des Roboters, da ein kleiner Unterschied in Pixeln zu großen Unterschieden in Millimetern führen kann. Dieser Fehler ist in gerader Linie zur Kamera häufiger zu beobachten, während die andere Richtung orthogonal zum ersten Fehler, minimal und eher um eine feste Größenordnung schwankt. Also muss Q_t zwei Fehler im Bezug zur Messung beinhalten. Der erste Fehler lässt sich über einen Winkelfehler im Bezug zur Kameratranslation beschreiben. Das Prinzip dieses Fehlers lässt sich in Abb. 9.1 ablesen.

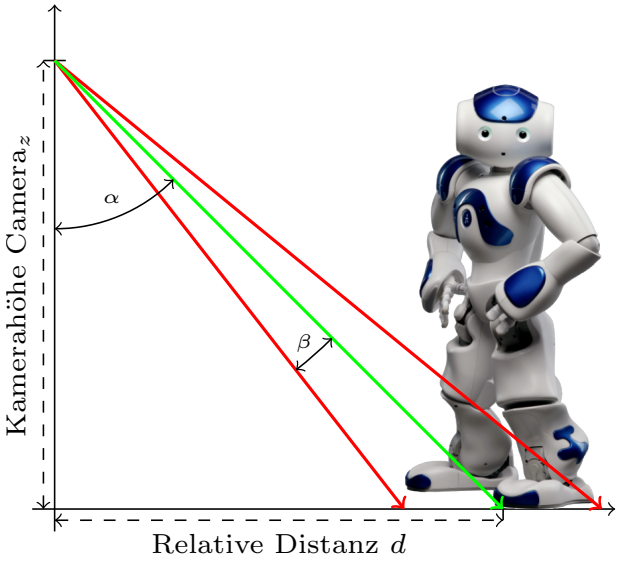


Abbildung 9.1 Illustration des Winkelfehlers β vom ursprünglichen Winkel α, gegeben einer Distanz d und der Kamerahöhe Camera$_z$.

Dabei lässt sich der Winkel α durch Winkelsätze berechnen, indem man die Messung in Weltkoordinaten projiziert. Dadurch erhält man einen Punkt μ (in Abb. 9.1 der untere Endpunkt der grünen Linie), der zu der Kameraposition $\begin{pmatrix}\text{Camera}_x\\ \text{Camera}_y\end{pmatrix}$ den Abstand d und einen Höhenunterschied im Sinne von Camera_z besitzt. Die Kameraposition und -höhe sind durch die Kalibrierung gegeben. Dann ergibt sich die Formel:

$$\alpha = \tan^{-1}\left(\frac{d}{\text{Camera}_z}\right) \tag{9.5}$$

Auf diesen Winkel α kann man dann den Fehler β, der in dieser Anwendung als $1,5°$ angenommen wird, in beide Richtungen anwenden. Durch diese neuen Winkel lässt sich dann der Winkelsatz in die umgekehrte Richtung anwenden, um neue Distanzen zu berechnen, in denen die äußeren Fehlerpunkte für den ersten Fall liegen.
Der zweite Fehlerfall lässt sich dann über den um 90° rotierten und normierten Kamera-zu-Perzept-Vektor berechnen, der auf einer Länge von 120 mm in beide Richtungen auf das Perzept angewandt wird. Die insgesamt vier Fehlerextrema sind beispielhaft in Abb. 9.2 dargestellt.

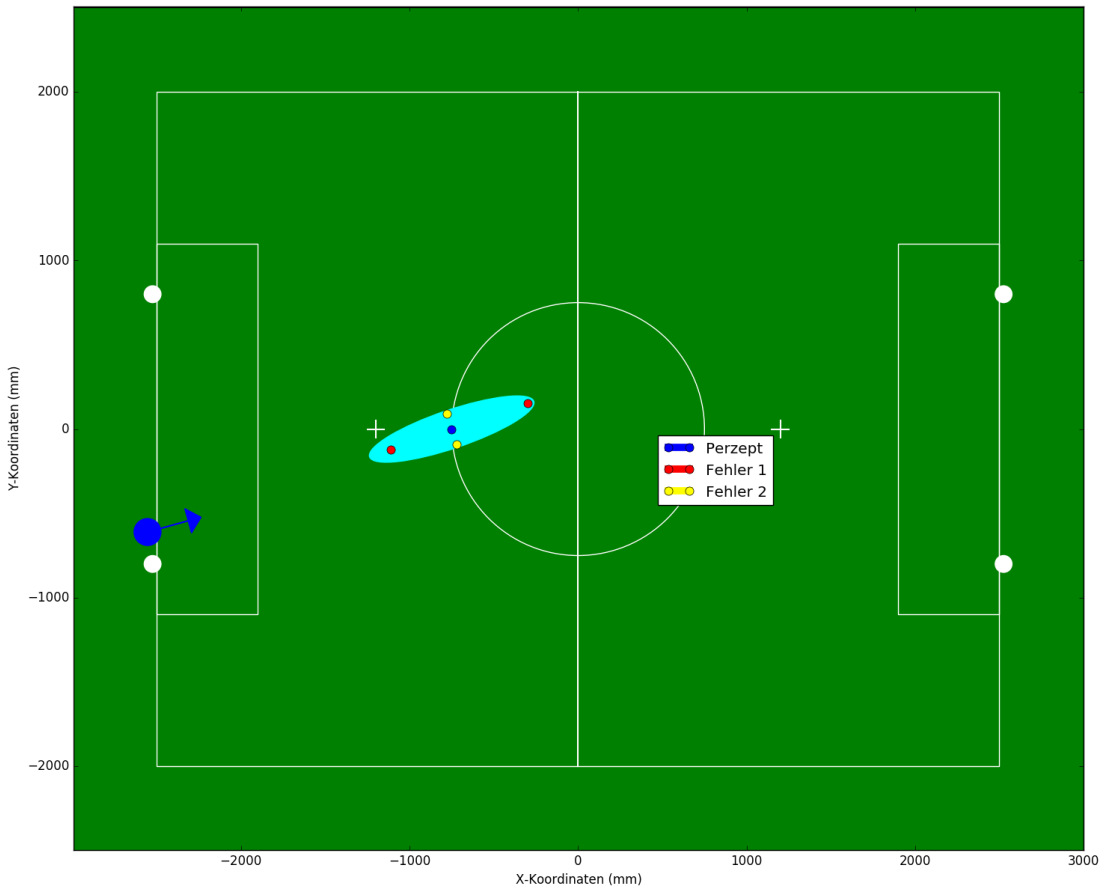


Abbildung 9.2 Vollständige Darstellung der beispielhaft berechneten Fehlerextrema um ein Perzept. Die Ellipse soll näherungsweise die resultierende Kovarianz darstellen, während der blaue Kreis mit dem Pfeil die Kameratranslation und deren Ausrichtung beschreibt.

Die Formeln zur Berechnung der Extrema in Weltkoordinaten sind:

$$d_{direction} = normalized(\text{Camera} \rightarrow \mu) \tag{9.6}$$

$$y_{1,2} = \tan(\alpha \pm 1,5) \cdot \text{Camera}_z \cdot d_{direction} + \begin{pmatrix} \text{Camera}_x \\ \text{Camera}_y \end{pmatrix} \tag{9.7}$$

$$y_{3,4} = \mu \pm 120 \cdot RotationMatrix(90°) \cdot d_{direction} \tag{9.8}$$

Die berechneten Weltkoordinaten y_i werden dann in das Bild transformiert und in der Form von Bildpunkten x_i mit ihrer Distanz zur Messung z_t gemittelt, um die Kovarianzmatrix Q_t wie folgt zu formen:

$$Q_t = \frac{1}{2} \sum_{i=1}^{n} (x_i - z_t)(x_i - z_t)^T \tag{9.9}$$

Eine angenehme Eigenschaft des *Unscented Kalman-Filters* ist, dass hiermit keine weiteren Variablen mehr spezifiziert werden müssen, da keine Jakobimatrizen oder weiteren Parameter benötigt werden und alle weiteren Werte sich durch Gleichungsanwendungen im Verlauf der Filterung ergeben.

9.2 Perzeptintegration

Um alle Schätzungen zu verwalten, wird der *HypothesesManager* benutzt. Diese Klasse nimmt alle Perzepte des *RobotDector*s entgegen und wandelt diese in Schätzungen um. Initial ist dieses Schätzproblem recht einfach, da die Liste der Schätzungen leer ist und in der Perzeptliste keine Doppelungen vorhanden sind. Dadurch kann man für jedes Perzept ein *Unscented Kalman-Filter* initialisieren, in dem man den Bildpunkt über Formel 7.16 in Weltkoordinaten transformiert und somit eine initiale Schätzung besitzt. Diese Schätzungen kann man durch das Zeichnen der Kovarianzen in der Feldansicht visualisieren, wie in Abb. 9.3 abgebildet.
Bei der weiteren Anwendung wird das Problem jedoch wesentlich komplexer. Mit einer bereits existierenden Liste an Schätzungen und einer neuen Liste an Perzepten kann man nicht grundsätzlich annehmen, dass die Reihenfolge der Perzepte auch der Reihenfolge der Schätzungen entspricht. Durch weitere detektierte Roboter können neue Elemente der Liste hinzugefügt werden oder durch fehlgeschlagenes Tracking können Elemente fehlen. Daher ist es ratsam ein komplexeres Modell für die Perzeptintegration zu verwenden, nämlich das *Gaussian Mixture Model*. Wie in Kapitel 4.4.1 erwähnt, ist ein *Gaussian Mixture Model* eine Verteilung über mehrere Normalverteilungen. Durch die Liste der Schätzungen in Form von *Unscented Kalman-Filtern*, welche ihre Zustände über einen Mittelwert μ_t und eine Varianz Σ_t beschreiben, lassen sich diese auch als *Gaussian Mixture Model* interpretieren. Anschließend kann man über das *Gaussian Mixture Model* mit Formel 4.13 berechnen, zu welcher Schätzung die Messung am wahrscheinlichsten gehört. Da das *Gaussian Mixture Model* von *OpenCV* nur als Lernverfahren anwendbar ist, wurde die Implementierung aus dem B-Human-Framework be-

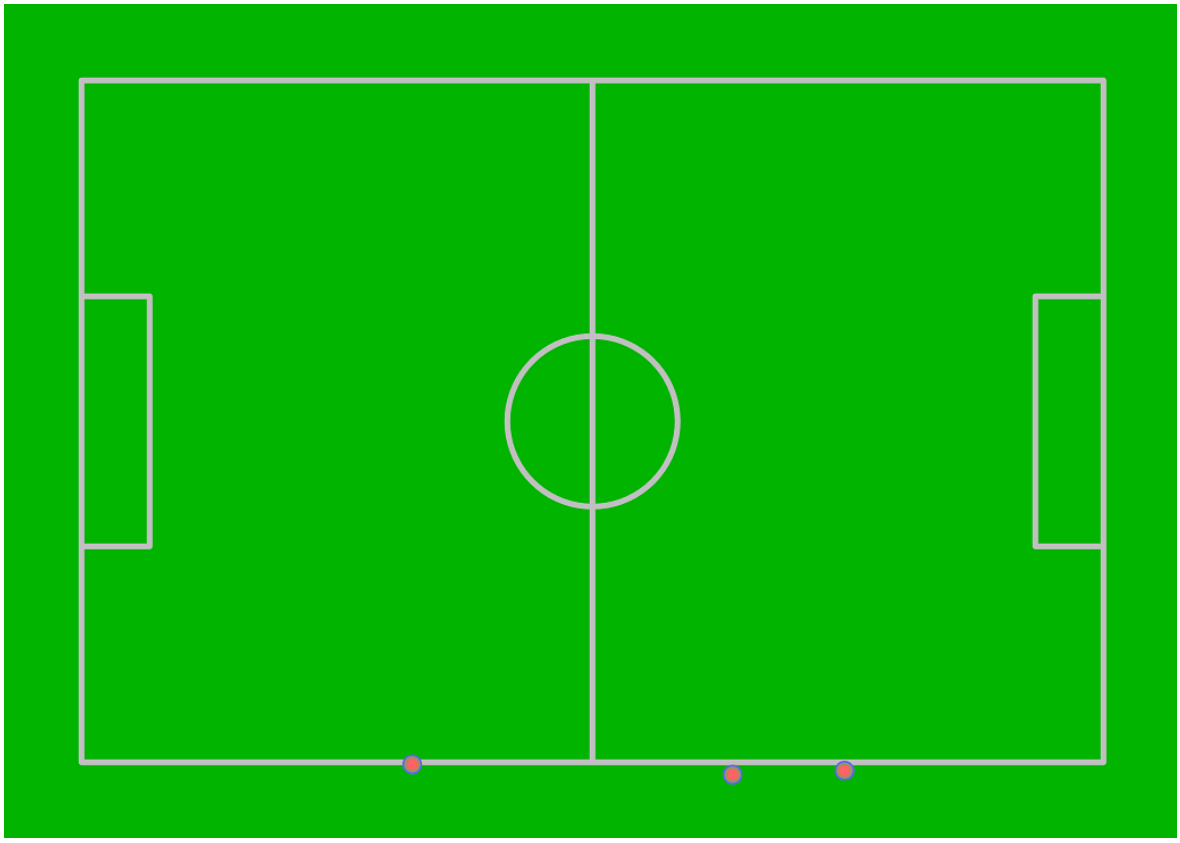


Abbildung 9.3 Visualisierung der Schätzung nach der ersten Detektion.

nutzt. Die Verteilungseigenschaften der Filter werden dem *Gaussian Mixture Model* übermittelt und dann wird für jede Messung die maximale normierte Wahrscheinlichkeit berechnet. Eine Messung wird einer Verteilung zugeordnet, wenn die Wahrscheinlichkeit größer gleich 0,4 ist. Ist dies nicht der Fall, wird die Messung für die Initialisierung einer neuen Schätzung benutzt. Wurden alle Messungen zugeordnet, wird für jede Schätzung die vollständige *Unscented Kalman-Filter* Routine durchlaufen, bestehend aus einem Dynamikschritt und einem Messschritt, falls eine Messung für diese Schätzung vorliegt. Diese Routine ermöglicht eine fortlaufende Schätzung über mehrere Roboterpositionen, wie in Abb. 9.4 dargestellt.
Wie in Abb. 9.4 zu sehen, ist die Unsicherheit bei weiter entfernten Schätzungen durch die Modellierung des Messfehlers größer. Dabei besitzt die Schätzung rechts außen mehr Unsicherheit, da vom Filter aus dort größere Schwankungen in der Projektion in die Schätzung integriert wurden. Dieses Phänomen ist die logische Konsequenz der Verwendung des *Unscented Kalman-Filters*, da für die Berechnung der Unsicherheit Σ_t die Sigmapunkte und ihre Differenzen zu μ_t in gemittelter Form verwendet werden. Jedoch konvergiert die Unsicherheit nach einer Reihe von gleichmäßigen Detektionen. Das aktuelle Bild für die Schätzung von Abb. 9.4 ist in Abb. 9.5 zu sehen.

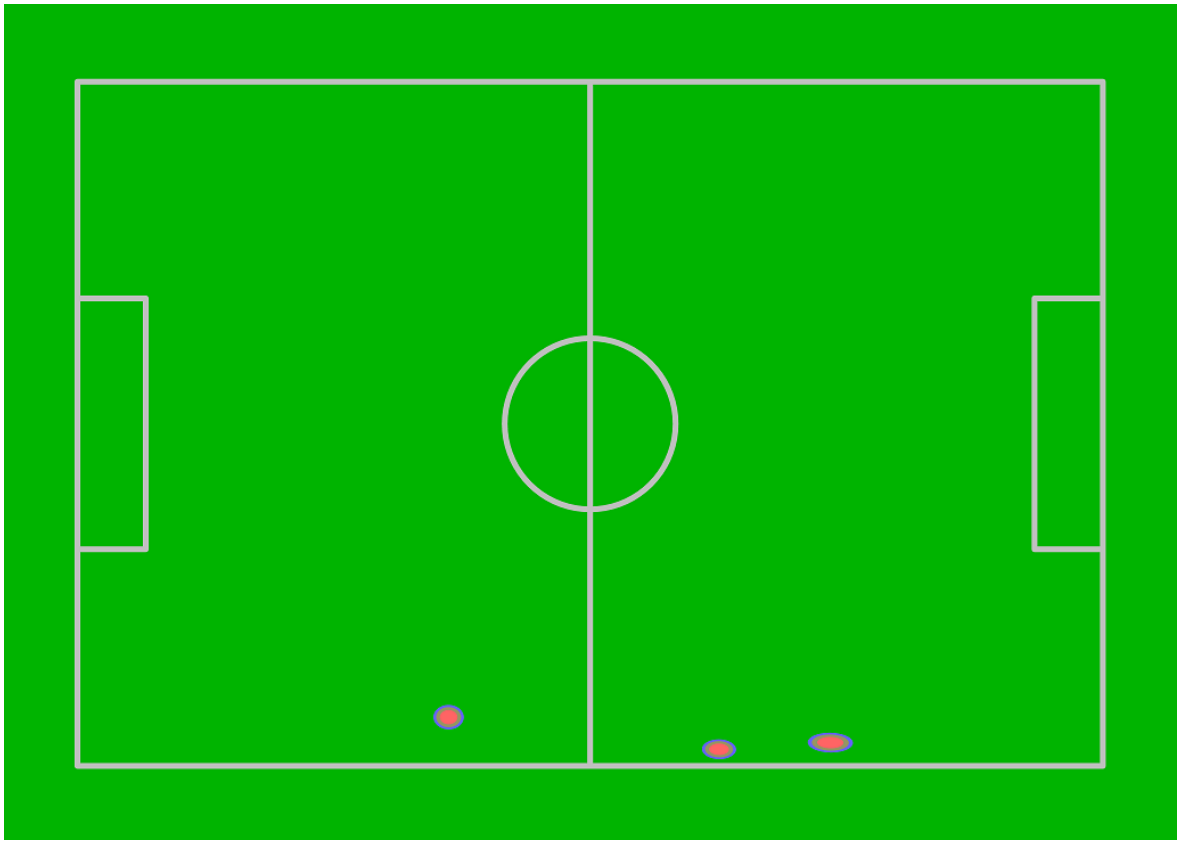


Abbildung 9.4 Visualisierung der Schätzung nach mehreren Detektionsdurchläufen.

9.3 Weitere Annahmen und Verbesserungen

Der *HypothesesManager* hat im Laufe der Entwicklung auch geholfen, falsche Perzepte auszusortieren. Damit ließen sich Unzulänglichkeiten vorheriger Roboterdetektoren ausgleichen. Einige Annahmen, die immer noch sinnvoll sind, werden hier umgesetzt, so dass möglichst valide Schätzungen entstehen:

1. Korrekt erkannte Roboter werden meist kontinuierlich erkannt: Diese Annahme sorgt dafür, dass Roboter, die zu lange nicht erkannt wurden, aus der Liste der Schätzungen entfernt werden. Die Überprüfung dieser Eigenschaft ist ein Überbleibsel von vorherigen Robotererkennern, bei denen keine Unterscheidung zwischen Tracking bekannter Roboter und Erfassen unbekannter Roboter gemacht wurde. Dennoch ist diese Annahme immer noch valide, da es sein kann, dass das Tracking fehlschlägt und in einem kurzen Zeitraum keine Messung entstehen kann, die die Schätzung bestätigt. Deshalb werden Schätzungen, die proportional zu ihrer vorherigen Detektionsdauer nicht mehr detektiert wurden oder deren Schätzung aus dem Feld driftet, gelöscht.

2. Durch Ausreißer können Detektionen nicht als zusammengehörig erkannt werden: Wenn über einen langen Zeitraum eine präzise Schätzung eines Roboters entsteht und dann ein Ausreißer detektiert wird, kann es sein, dass bei der Sensorintegration über das

Gaussian Mixture Model die Messung nicht korrekt zugeordnet wird. Dadurch entstehen benachbarte Schätzungen, die zusammengeführt werden müssen. Um dies zu erreichen wird eine anschließende Routine ausgeführt, welche die Bhattacharyya-Distanz über zwei Verteilungen $\mathcal{N}(\mu_1, \Sigma_1)$ und $\mathcal{N}(\mu_2, \Sigma_2)$ in Anlehnung an Choi und Lee [2003, S. 1704] verwendet.

$$\Sigma = \frac{\Sigma_1 + \Sigma_2}{2} \tag{9.10}$$

$$D_B = \frac{1}{8}(\mu_1 - \mu_2)^T \Sigma^{-1} (\mu_1 - \mu_2) + \frac{1}{2} \ln \left(\frac{\det \Sigma}{\sqrt{\det \Sigma_1 \det \Sigma_2}} \right) \tag{9.11}$$

Jede der Schätzungen wird über diese Metrik miteinander verglichen. Sollte die Distanz zwischen zwei Verteilungen kleiner als zwei sein, werden diese durch Mittelung zusammengeführt. Dieses Verfahren diente auch bei älteren Versionen des Robotererkenners dazu, zwei Perzepte, die zum selben Roboter im selben Bild entstanden sind, auf der Schätzungsebene zusammenzuführen. Ebenso findet es Anwendung, um, wenn durch ein zu kleinen ROI simultan Ober- und Unterkörper erfasst werden, diese nach mehreren Detektionszyklen konvergieren zu lassen.

3. Es existieren nicht mehr als 10 Roboter auf dem Feld: Durch die Regeln [RoboCup Technical Committee, 2017] beträgt die Spieleranzahl 5 Roboter pro Team. Dies ist eine Maßnahme, um die Entstehung zu vieler Schätzungen zu vermeiden. In dem Fall werden die nach Bhattacharyya-Distanz am dichtesten beieinanderliegenden Verteilungen fusioniert.

Durch die Anwendung dieser Modellierung entstehen die Referenzdaten, die das System, wie in Abb. 9.5 dargestellt, auch direkt ausgibt. Dort sind die Regionen des Trackings als rote Rechtecke, die letzten Detektionspunkte als rote Kreise und die in das Bild projizierten Zustände als blaue Ziffern eingezeichnet, korrespondierend zu ihrer Schätzungsnummer. Alternativ wird von dem System wie in Kapitel 9.4 eine CSV-Datei erstellt.

9.3.1 Diskussion des Aufbaus

Viele Probleme in der Robotik lassen sich auf vielfältige Weise modellieren. Daher ist es auch sinnvoll, mögliche Änderungen für dieses System zu diskutieren. Die verschiedensten Arten von Kalman-Filtern finden in der Robotik großen Zuspruch, aber auch Partikelfilter werden häufig benutzt [vgl. Kwok und Fox, 2005, Kapitel 2]. Die Entscheidung für den *Unscented Kalman-Filter* fiel auf Grund der vielen Adaptionsmöglichkeiten und der Vertrautheit mit diesem Filter. Jedoch bieten auch diese vielen Adaptionsmöglichkeiten ebenso viel Diskussionsmaterial. Der erste Aspekt ist die Wahl des Zustands μ_t als Schätzung. Hier wird die reine Position in Weltkoordinaten geschätzt, weshalb sich aus dem Zustand kein Dynamikmodell g zur Modellierung einer Bewegung ergibt. Jedoch lässt sich die Differenz der Projektionen über einen gewissen Zeitraum als Bewegung interpretieren und als solche auch in den *Unscen-*

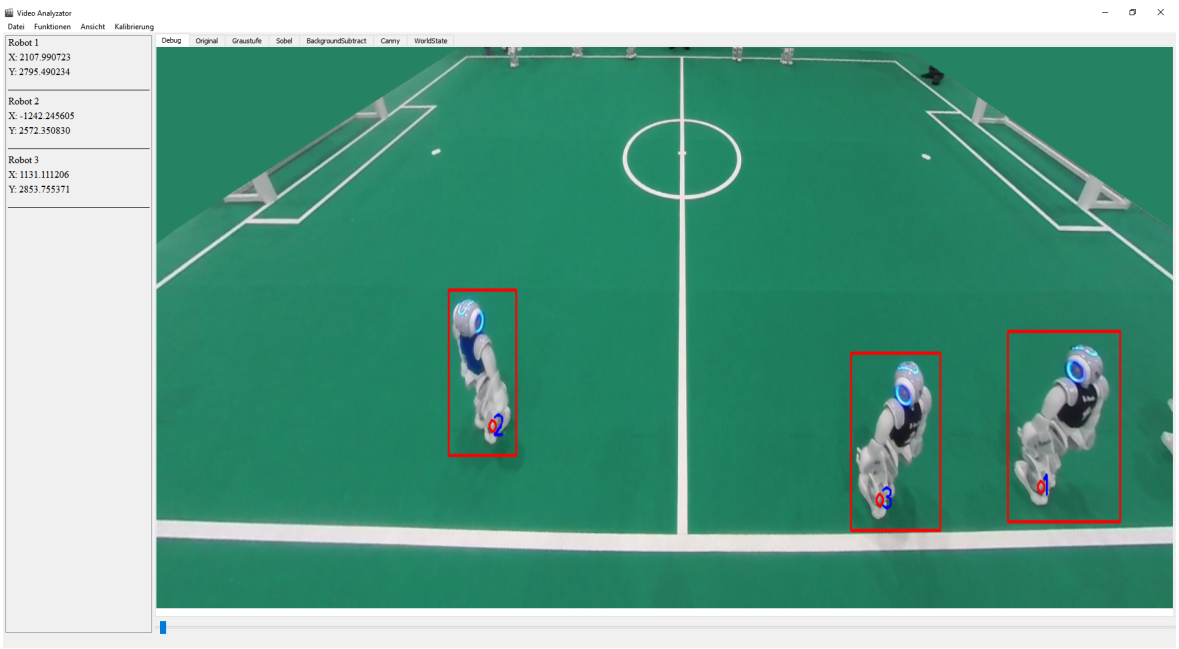


Abbildung 9.5 Visualisierung der Gesamtschätzung in einem *QWidget* auf der linken Seite. Im Bild sind die ROIs, die letzten Detektionen und die in das Bild projizierten Schätzungen zu sehen.

ted Kalman-Filter integrieren. Dieser Ansatz erwies sich im Laufe der Entwicklung als wenig praktikabel, da vor allem an den Bildrändern recht geringe Unterschiede in der Bildposition große Unterschiede in der Projektion bewirken. Dadurch lässt sich eine Geschwindigkeit nur unzureichend schätzen. Ein Aspekt, der für die Entscheidung für eine Mittelung über Positionen spricht, ist die Beobachtung, dass sich die erfasste Roboterposition zwischen zwei aufeinander folgenden Bildern bei 30 bis 60 Bildern pro Sekunde nicht sehr stark unterscheidet. Für künftige Arbeiten können in diesem Gebiet aber weitere Untersuchungen stattfinden. Eine weitere mögliche Erweiterung des Zustands wäre mit der Erfassung von Rotation und/oder Lage möglich, wodurch komplexere Modelle denkbar wären. In einer anderen Untersuchung aus B-Human von Mühlenbrock und Laue [Im Erscheinen] wird gezeigt, dass über eine Roboterfußbetrachtung mit einer Konturdetektion und umliegender Grünverteilung eine Rotation des *NAO*s geschätzt werden kann. Würde man dadurch die Rotation von Robotern innerhalb dieses Systems messen, würden sich dadurch mehr Möglichkeiten im *Unscented Kalman-Filter* ergeben. Ähnlich könnte man durch eine Lageschätzung, ob der Roboter aufrecht steht oder auf dem Boden liegt, weitere Modellierungen in den *Unscented Kalman-Filter* integrieren und damit eine höhere Präzision erreichen.
Weitere Untersuchungen sind in dem Bereich durchaus interessant, da, wie in der Reflexion des Tracking-Problems von Kwok und Fox [2005], im RoboCup die folgenden Faktoren entscheidend sind und im Kontext dieser Arbeit beantwortet wurden:

- Nichtlineare Bewegung des Beobachters: Der grundlegende Akteur in dieser Arbeit ist die Kamera, welche als primäre Annahme in dieser Arbeit keine Eigenbewegung besitzt.

Diese Annahme vereinfacht das Tracking-Problem deutlich, da nur Bewegungen des Zielobjekts modelliert werden müssen.

- Physische Interaktion zwischen Ziel und Umgebung: Die Vorhersage von Roboterbewegungen ist ähnlich nichtlinear wie die Bewegung des Balles, da Bewegungen durch Fremdeinfluss entstehen können. Zusätzlich können die *NAO*s in der SPL auch selbstständig ihre Bewegung ändern. Daher wird auf die konkrete Vorhersage in dieser Arbeit verzichtet, aber dafür eine Unsicherheit modelliert. In weiteren Arbeiten sollte hier eine ausführliche Untersuchung stattfinden, da durch eine präzise Vorhersage von Bewegungen und Wechselwirkungen in der Modellierung die Qualität des Trackings deutlich erhöht werden kann.

- Physische Interaktion zwischen Beobachter und Ziel: Ist in dieser Arbeit nicht gegeben, da die Kamera keinen Einfluss auf den Roboter hat (zumindest nicht direkt und nicht geplant, falls eine Fehllokalisierung durch das Aufstellen der Kamera entstehen sollte).

- Unpräzise Messung und begrenzte Rechenzeit: Das Problem der unpräzisen Messungen gilt es, wie bei der Roboterdektion erwähnt, zu evaluieren, aber verspricht eine Verbesserung, da eine *GoPro* eine wesentlich höhere Auflösung als der *NAO* besitzt. Das Problem der begrenzten Rechenzeit wird dabei umgangen, indem das Tracking nicht auf dem *NAO* oder gar dem AIBO stattfindet, sondern offline auf einem Rechner der eigenen Wahl.

Diese Faktoren zeigen, dass durchaus komplexere Ansätze verfolgt werden könnten, wie zum Beispiel Rao-Blackwellisierung, um mehrere Dynamikmodelle anwenden und damit Nichtlinearität kompensieren zu können. Dabei ist es besonders interessant, dass in diesem Kontext die Eigenbewegung und Beschränkung der Rechenzeit wegfallen.
Ein anderer interessanter Ansatz ist von Jochmann u. a. [2012], bei dem die Lokalisierung über einen *Unscented Kalman-Filter* mit einem *belief* in Form eines *Gaussian Mixture Models* realisiert wird. Wie in Kapitel 3.4 erwähnt, wird dort das Kidnapped Robot Problem für den *NAO* behandelt, welches in diesem System auch vorhanden ist, wenn man die Interaktion der Schiedsrichter mit den Roboter berücksichtigen und modellieren möchte. Dabei ist aber eine andere Grundannahme vorhanden, bei der man den Roboter eindeutig wieder identifizieren müsste, was aus der Perspektive dieses Systems nicht gegeben ist, während in der Entwicklung von Jochmann u. a. [2012] der *NAO* sich nur selbst lokalisieren will und nicht das ganze Team.
Trotz der vielen Erweiterungsmöglichkeiten ist abschließend zu sagen, dass der jetzige Ansatz in der Entwicklung genügend Präzision aufwies, um anwendbar zu erscheinen.

9.4 Speichern der Referenzdaten

Die Berechnung der einzelnen Daten wird für jedes einzelne Bild durchgeführt und ebenso die Datenverwaltung, um eine sortierte Liste an Referenzdaten zu erhalten. Zu diesem Zweck wird vom *HypothesesManager* eine Liste von Listen verwaltet, die diese Daten beinhaltet. Dieses Konstrukt lässt sich als eine Tabelle vorstellen, die die Daten eines jeden Roboters in eine dafür vorgesehene Zeile schreibt. Dabei stehen die neueren Daten des Roboters in der Spalte, die sich am weitesten rechts befindet. Dabei müssen Werte, die sich in der selben Spalte befinden, nicht aus dem selben Bild des Videos berechnet worden sein. Das Prinzip ist in Tab. 9.1 visualisiert.

Roboter\Zeitschritt	1	2	3	4	5	6	7	8	9
1: μ_1	μ_{1_1}	μ_{1_2}	μ_{1_3}	μ_{1_4}	μ_{1_5}	μ_{1_6}			
2: μ_2	μ_{2_1}	μ_{2_2}	μ_{2_3}	μ_{2_4}					
3: μ_3	μ_{3_1}	μ_{3_2}	μ_{3_3}	μ_{3_4}	μ_{3_5}	μ_{3_6}	μ_{3_7}	μ_{3_8}	μ_{3_9}
4: μ_4	μ_{4_1}	μ_{4_2}	μ_{4_3}						
5: μ_5	μ_{5_1}	μ_{5_2}	μ_{5_3}	μ_{5_4}	μ_{5_5}	μ_{5_6}			

Tabelle 9.1 Beispielhafter Aufbau des Datencontainers der Referenzdaten.

Das Konstrukt wird nach jedem Anwendungszyklus des *HypothesesManager*s aktualisiert. Dies geschieht, indem jedem *Unscented Kalman-Filter* eine Zeilennummer zugewiesen wird. Alle Filter, die eine solche Nummer besitzen, können dadurch ihre Zeile identifizieren und die Daten dort hinein schreiben und setzten in einer Prüfliste ihre Zeile als beschrieben ein. Besitzt ein Filter keine Nummer, so muss mehr Aufwand betrieben werden, da über alle Zeilen iteriert wird und überprüft wird, ob die Zeile schon beschrieben wurde. Alle Zeilen, die bereits beschrieben wurden, sollen dem neuen Filter nicht zugeordnet werden. Bei einer unbeschriebenen Zeile, deren letzter Eintrag näher als 50 cm ist, wird der Filter dieser Zeile zugeordnet. Konnte keine geeignete Zeile auf diese Weise gefunden werden, wird für den Filter eine neue Zeile erstellt und ihm diese zugeordnet.
Über die Benutzeroberfläche kann die Erstellung einer CSV-Datei zum Erhalt aller generierten Referenzdaten angefordert werden. Das System öffnet dadurch eine CSV-Datei und schreibt in die erste Zeile die relevanten Kalibrierungsdaten (Kameraposition und Ausrichtung) und dann zeilenweise die Daten des Referenzdatencontainers. Dadurch erhält man ein Datenformat, was man auch für Analyse-Zwecke automatisiert auslesen kann. Für die Visualisierung wurde ein Python-Programm entwickelt, welches über Kommandozeilenparameter die CSV-Datei sowie Flags zum Spiegeln der Koordinaten oder Skalierung des Feldes entgegennimmt und eine Zeichnung, wie in Abb. 9.6 dargestellt, visualisiert. Die Kameraposition und -ausrichtung sind als blauer Kreis mit Pfeil gekennzeichnet. Die Trajektorien aus jeder Zeile werden einer Farbe zugeordnet und als Funktionsverlauf im zweidimensionalen Raum eingezeichnet (wobei ein einzelner Punkt in einer Zeile zwar in der Legende vermerkt wird, aber nicht zu sehen ist). Wenn die CSV-Datei mehr als neun Roboterzeilen besitzt, können sich Farben wiederholen.

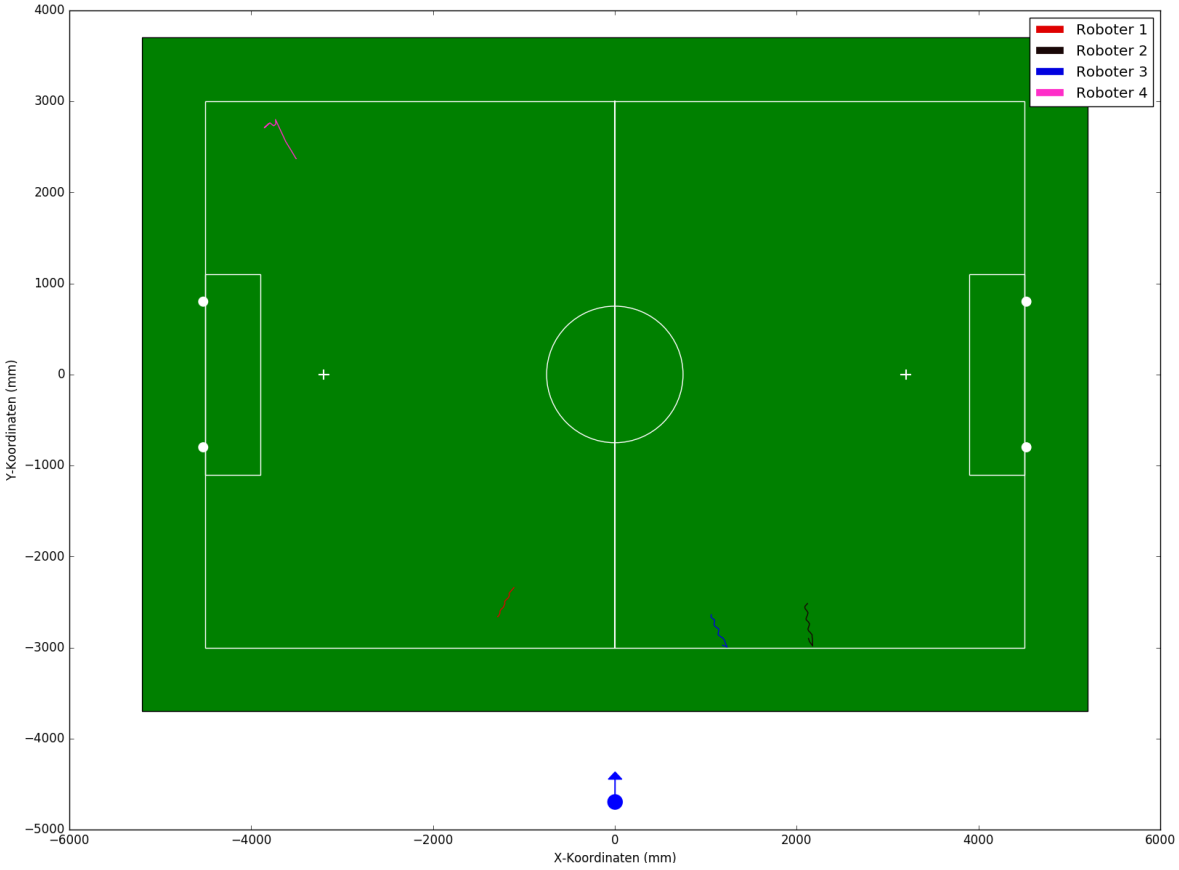


Abbildung 9.6 Beispielhafte Ausgabe des in Python geschriebenen Visualisierungsprogrammes.

Kapitel 10

Evaluation

In diesem Kapitel werden grundlegende Eigenschaften des vorgestellten Systems geprüft. Für jede dieser Eigenschaften gibt es einen eigenen Abschnitt, in dem der Versuchsaufbau und die Ergebnisse dargelegt werden.

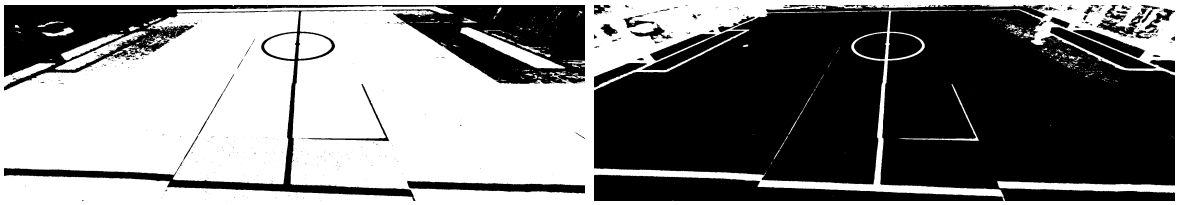
10.1 Vorwort zu der Evaluation

Das Ziel des entwickelten Systems sollte seine Anwendbarkeit im Kontext des RoboCups sein. Dies bedeutet, dass die meisten Tests auf dem für die SPL definierten Feld vom RoboCup Technical Committee [2017] statt finden müssten. Zu diesem Zweck wurden einige Videos für die Evaluation während eines B-Human Testspiels aufgenommen. Dadurch konnte eine Anforderung des Systems an die Videos identifiziert werden. Denn die Szene aus Abb. 10.1 lässt sich nach der Farbklassifizierung, wie in Kapitel 6 vorgestellt, nicht analysieren, selbst wenn man für diese Szene eine darauf zugeschnittene Parametrisierung verwendet.



Abbildung 10.1 Ein unverändertes Bild aus den nicht analysierbaren Testspielvideos.

Das Problem, abgesehen von den Aufbaufehlern (Lücken zwischen den Teppichstücken), ist die Grünfläche, auf dem der Roboter steht. Selbst wenn das *Gaussian Mixture Model* nur zwei Farbklassen unter Benutzung eines jeden Pixels im bereits auf das Feld zurecht geschnittene Bild mit einer niedrigeren Mindestwahrscheinlichkeit der Klassenakzeptanz anwendet, so werden die zwei Klassen wie in Abb. 10.2 gelernt. Dies legt nahe, dass diese Teppichpixel tatsächlich näher an Weiß sind als an Grün, weshalb die menschliche Grüninterpretation eher dem Kontext geschuldet ist.



(a) Grün (b) Weiß

Abbildung 10.2 Das Resultat des speziell parametrisierten *Gaussian Mixture Model* auf dem Testspielvideo.

Um dies zu überprüfen, wurde mit dem *VLC media player* das Video auf dessen Grünanteil hin untersucht. Der Teppich enthält wie in Abb. 10.3 visualisiert sehr viele Grüntöne, wobei auch weiße Flächen einen Grünanteil besitzen.

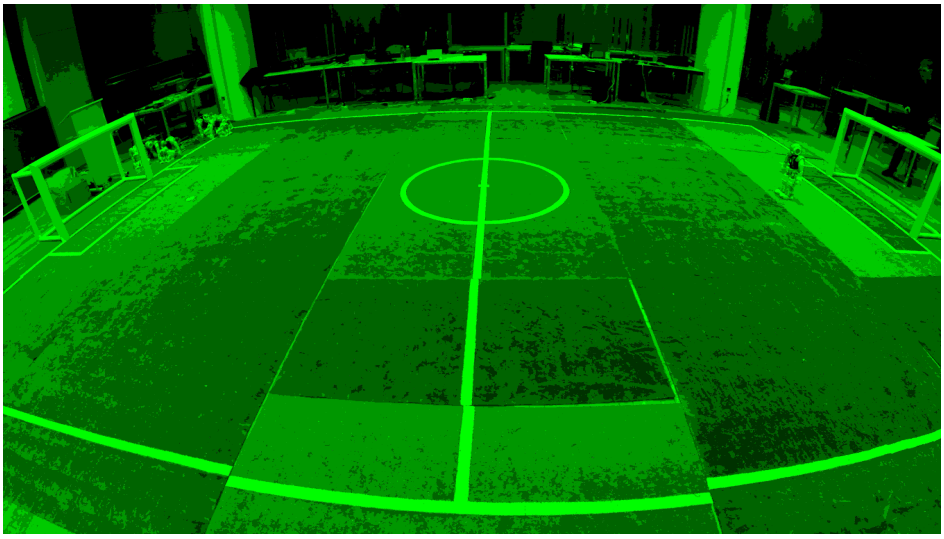


Abbildung 10.3 Die Farbtonuntersuchung im *VLC media player* nach Grün.

Über die manuelle Einstellung eines Grünanteil-Schwellwertes ließ sich ermitteln, bis zu welcher Obergrenze eine Farbe als Grün angesehen wird. Die beste Farbtrennung, ohne dass Grün verschwindet, ist in Abb. 10.4 abgebildet. Darin sind alle Pixel unterhalb eines Schwellwertes als Grün eingezeichnet und es ist ersichtlich, dass sich das Feld farblich nicht vollständig vom Roboter trennen lässt.
Dieses Phänomen lässt die Roboterdetektion fehlschlagen, da im Nicht-Grün-Bild die besagte Teppichfläche weiß und damit zum Roboter gehörig ist. Damit ist eine Voraussetzung zur

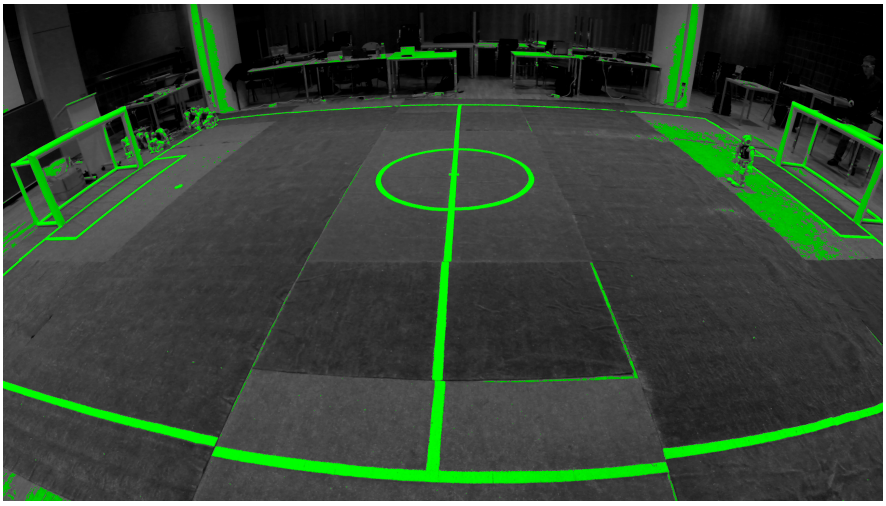
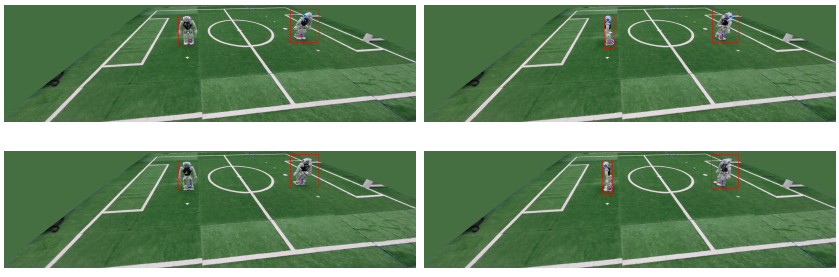


Abbildung 10.4 Der von *VLC media player* manuell ermittelte Farbschwellwert für Grüntrennung in angewandter Form.

Analyse von Videos mit dem System ein gleichmäßigeres Grün des Feldes. Aus diesem Grund wurden die meisten Versuche in den nachfolgenden Kapiteln auf dem Feld des Projektraums evaluiert, bei dem durch andere Beleuchtungsbedingungen ein gleichmäßigeres grün erzielt wird.

10.2 Ermittlung des Detektionsfehlers

Die Güte der Referenzdaten ist direkt von der Qualität der Kalibrierung abhängig, die diesem System vorliegt. Um zu ermitteln, wie gut die möglichen Kalibrierungen in diesem System anwendbar sind, wird der Detektionsfehler über fest definierte Roboterpositionen ermittelt und mit dem bereits von *OpenCV* errechneten Projektionsfehler verglichen. Der Versuch wurde auf einem Feld ausgeführt, welches nicht den Regeln [RoboCup Technical Committee, 2017] entspricht, aber dessen Ausmaße bekannt sind. Die für den Versuch benutzten zehn Positionen befanden sich auf X-Koordinatenhöhe der Strafstoßpunkte. Dabei sind zwei der Positionen direkt auf ihnen befindlich, wobei sich die anderen jeweils 50 cm und 100 cm vor- und hinter ihnen in Y-Richtung befinden. Während des Versuchs wurden die Roboter in vier verschiedenen Orientierungen gefilmt und dasselbe mit liegenden Robotern ausprobiert, wie in Abb. 10.5 dargestellt.



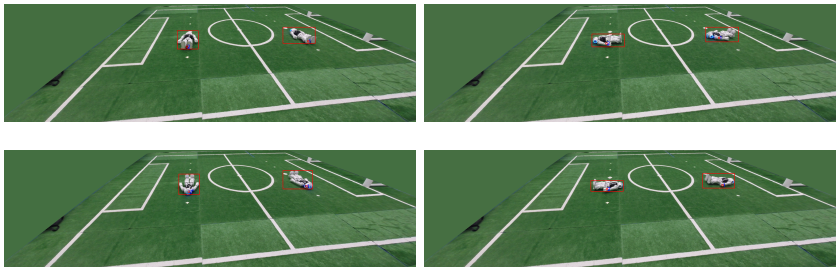


Abbildung 10.5 Der verwendete Aufbau zur Ermittlung des Detektionsfehlers bezüglich der Position mit den verwendeten Ausrichtungen der Roboter. Hier exemplarisch 50 cm vor den Strafstoßpunkten.

Die ermittelten Punkte sind als Punktwolken in Abb. 10.6 mit der durch Kalibrierung ermittelten Kamerapose (in blau) und der tatsächlichen Kamerapose (in rot) visualisiert. Jede der gewünschten Positionen wird durch die Legende nummeriert und kann dadurch statistisch untersucht werden. Alle anderen Zielpositionen, abgesehen von den Strafstoßpunkten, wurden wie Torpfosten hervorgehoben.

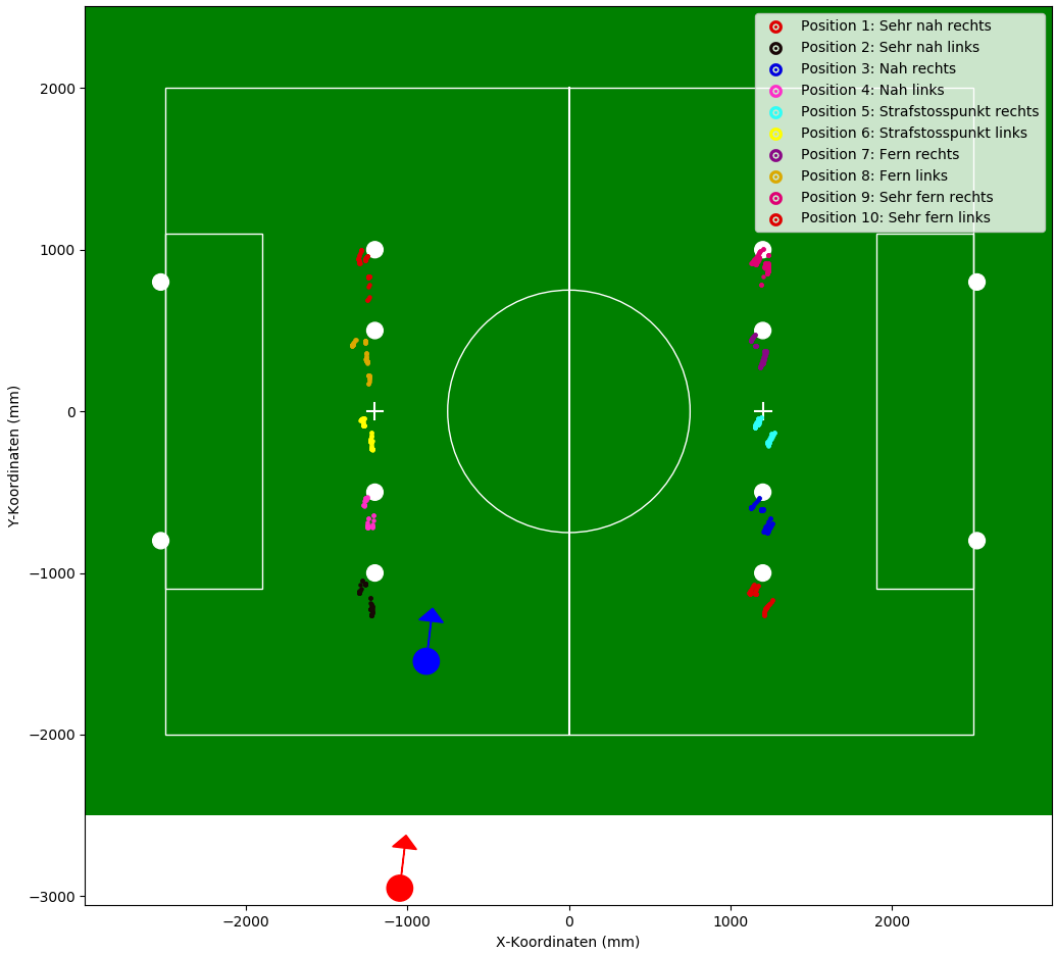


Abbildung 10.6 Die Gesamtvisualisierung der ermittelten Fußpunkte über alle Roboterpositionen, -rotationen und -lagen. Die Kamerapose der Kalibrierung ist in blau und die tatsächliche in rot eingezeichnet.

Bei diesem Aufbau wurden zwei Kamerapositionen ermittelt:

$$\text{Die gemessene Position:} \qquad C_t = \begin{pmatrix} 104,6 \text{ cm} \\ 295 \text{ cm} \\ 141 \text{ cm} \end{pmatrix} \tag{10.1}$$

$$\text{Die Position nach Kalibrierung:} \qquad \bar{C}_t = \begin{pmatrix} 88,19 \text{ cm} \\ 154,62 \text{ cm} \\ 84,72 \text{ cm} \end{pmatrix} \tag{10.2}$$

Damit liegt im dreidimensionalen Raum ein Abstand von $152,129$ cm zwischen der tatsächlichen Position C_t und der kalibrierten Schätzung $\bar{C}_t$. An der Visualisierung von Abb. 10.6 ist zu erkennen, dass beide Positionen im nahezu gleichen Winkel zum Ursprung stehen. Damit wurde nachgewiesen, dass die Projektionen durch *OpenCV* mit extrinsischen Parametern funktionieren, die nicht absolut die realen Parameter erreichen, aber dennoch die Ausrichtung richtig erfassen.
Während der Auswertung wurde der Parameter der Anzahl der Erodierungen angepasst, damit die initiale Roboterdetektion die Roboter auch auf größere Distanz erkennt. Trotz der Parametrisierung wurde ein Roboter in einer Position nicht erkannt. Diese Szene ist in Abb. 10.7 dargestellt. Für den linken Roboter war der Schwarz-Anteil in seiner Region nicht groß genug. Ansonsten wurden alle anderen Daten über jeweils 20 Bilder ermittelt.



Abbildung 10.7 Der einzige Fall, in dem keine Daten von einem Roboter erzeugt werden konnten.

Über alle gesammelten Daten wurden die folgenden Daten berechnet:

- T: Der erwünschte Detektionswert im Sinne der realen Position.
- μ_G: Der Mittelpunkt einer Normalverteilung über alle Punkte einer Position.
- μ_S: Der Mittelpunkt einer Normalverteilung über die Punkte der stehenden Roboter einer Position.
- μ_F: Der Mittelpunkt einer Normalverteilung über die Punkte der liegenden Roboter einer Position.

- RMS_G: Der Root-Mean-Squared-Error über alle Punkte einer Position.
- RMS_S: Der Root-Mean-Squared-Error über die Punkte der stehenden Roboter einer Position.
- RMS_F: Der Root-Mean-Squared-Error über die Punkte der liegenden Roboter einer Position.
- $RMS_{\Omega G}$: Der Root-Mean-Squared-Error über alle Punkte.
- $RMS_{\Omega S}$: Der Root-Mean-Squared-Error über alle Punkte der stehenden Roboter.
- $RMS_{\Omega F}$: Der Root-Mean-Squared-Error über alle Punkte der liegenden Roboter.

Position	T	μ_G	μ_S	μ_F	RMS_G	RMS_S	RMS_F
1	$\begin{pmatrix}1200\\-1000\end{pmatrix}$	$\begin{pmatrix}1219.58\\-1076.23\end{pmatrix}$	$\begin{pmatrix}1259.08\\-982.29\end{pmatrix}$	$\begin{pmatrix}1180.09\\-1170.17\end{pmatrix}$	142.81	77.45	186.52
2	$\begin{pmatrix}-1200\\-1000\end{pmatrix}$	$\begin{pmatrix}-1234.02\\1067.51\end{pmatrix}$	$\begin{pmatrix}-1217.93\\-976.4\end{pmatrix}$	$\begin{pmatrix}-1250.1\\-1158.62\end{pmatrix}$	134.38	42.06	185.33
3	$\begin{pmatrix}1200\\-500\end{pmatrix}$	$\begin{pmatrix}1238.26\\-553.48\end{pmatrix}$	$\begin{pmatrix}1280.60\\-443.26\end{pmatrix}$	$\begin{pmatrix}1195.91\\-663.71\end{pmatrix}$	148.02	102.58	182.47
4	$\begin{pmatrix}-1200\\-500\end{pmatrix}$	$\begin{pmatrix}-1230.79\\-541.22\end{pmatrix}$	$\begin{pmatrix}-1216.42\\-449.91\end{pmatrix}$	$\begin{pmatrix}-1245.16\\-632.52\end{pmatrix}$	121.18	62.38	159.62
5	$\begin{pmatrix}1200\\0\end{pmatrix}$	$\begin{pmatrix}1247.82\\-8.04\end{pmatrix}$	$\begin{pmatrix}1293.72\\113.37\end{pmatrix}$	$\begin{pmatrix}1201.92\\-129.44\end{pmatrix}$	150.44	153.51	147.32
6	$\begin{pmatrix}-1200\\0\end{pmatrix}$	$\begin{pmatrix}-1235\\-10.49\end{pmatrix}$	$\begin{pmatrix}-1221.54\\110.78\end{pmatrix}$	$\begin{pmatrix}-1248.47\\-131.76\end{pmatrix}$	142.99	123.77	159.92
7	$\begin{pmatrix}1200\\500\end{pmatrix}$	$\begin{pmatrix}1225.43\\496.95\end{pmatrix}$	$\begin{pmatrix}1281.26\\627.93\end{pmatrix}$	$\begin{pmatrix}1171.74\\371.02\end{pmatrix}$	151.46	153.3	149.67
8	$\begin{pmatrix}-1200\\500\end{pmatrix}$	$\begin{pmatrix}-1251.12\\492.86\end{pmatrix}$	$\begin{pmatrix}-1230.7\\641.3\end{pmatrix}$	$\begin{pmatrix}-1271.54\\344.42\end{pmatrix}$	173.97	146.92	197.35
9	$\begin{pmatrix}1200\\1000\end{pmatrix}$	$\begin{pmatrix}1209.69\\956.56\end{pmatrix}$	$\begin{pmatrix}1223.21\\1020.27\end{pmatrix}$	$\begin{pmatrix}1198.87\\905.92\end{pmatrix}$	87.97	48.88	109.63
10	$\begin{pmatrix}-1200\\1000\end{pmatrix}$	$\begin{pmatrix}-1248.32\\1013.3\end{pmatrix}$	$\begin{pmatrix}-1237.3\\1098.96\end{pmatrix}$	$\begin{pmatrix}-1263.02\\899.08\end{pmatrix}$	164.85	178.48	144.69

Tabelle 10.1 Die statistische Auswertung der Detektionen mit bis auf zwei Nachkommastellen gerundeten Werten in Millimetern.

$$RMS_{\Omega G} = 142.72572237524287 \tag{10.3}$$

$$RMS_{\Omega S} = 118.23747393296182 \tag{10.4}$$

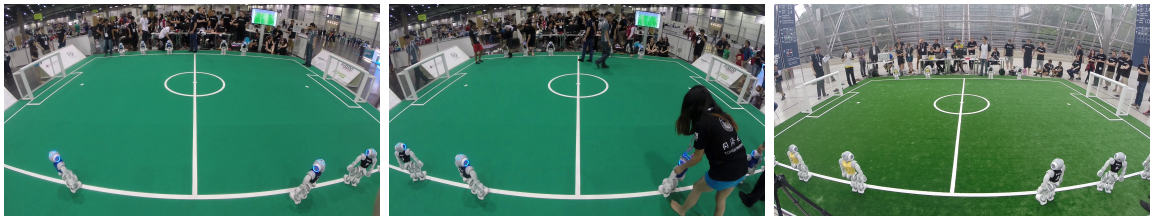
$$RMS_{\Omega F} = 163.514949953175 \tag{10.5}$$

Die ermittelten Werte zeigen, dass trotz der Detektion von liegenden Robotern, auf die das System mit den Annahmen in der Fußpunktdetektion nicht ausgelegt ist, eine ausreichende

Präzision besitzt. Durch den ermittelten Wert $RMS_{\Omega G}$ von nahezu 14,3 cm lässt sich aussagen, dass im Durchschnitt über alle Detektionen der ermittelte Wert nur 14,3 cm von dem tatsächlichen Wert abweicht.

10.3 Evaluation der initialen Spielerkennung

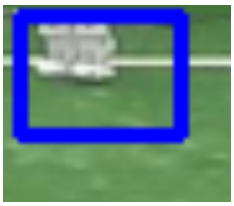
Der für dieses System vorgestellte Robotererkenner trifft einige Annahmen darüber, wie die extrahierten Regionen rund um Roboter auszusehen haben. Um abzuschätzen, wie realistisch all diese Annahmen sind, muss die initiale Roboterdetektion auf einem gesamten Spiel getestet werden. Hierzu wurden drei Videos mit jeweils 200 zu analysierenden Bildern erstellt, die aus den ursprünglichen Testvideos vom RoboCup 2016 in Leipzig generiert wurden. Zwei Videos stammen aus der Indoor-Competition und ein Video stammt aus der Outdoor-Competition. Die nicht ausgewerteten Anfangsbilder sind in Abb. 10.8 zu sehen.

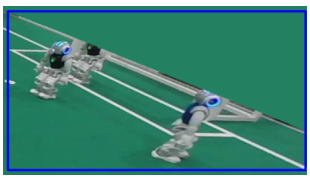
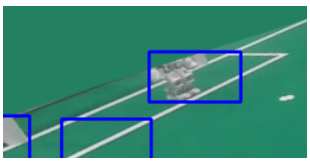
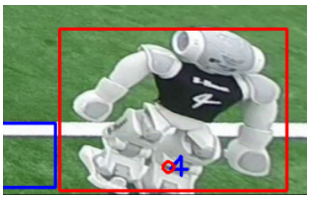
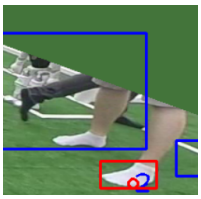
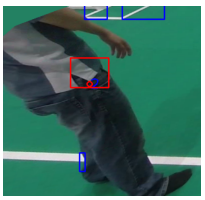


(a) Indoor 1 (b) Indoor 2 (c) Outdoor

Abbildung 10.8 Die Anfangsbilder der für diesen Versuch ausgewerteten Videos.

Jedes der anderen 200 Bilder wurde aus diesen Videos zufällig extrahiert und in die generierten Videos eingefügt. Die resultierenden Videos wurden einzeln durchlaufen und die Resultate der Detektion in den Unterabschnitten erfasst, wobei alle Gleitkommazahlen auf zwei Nachkommastellen gerundet sind. Die Kriterien, die bei der Auswertung benutzt wurden, sind in Tab. 10.2 dargestellt und beinhalten vom Autor erfasste Eigenschaften, Detektionsergebnisse und ermittelte statistische Werte.

Kriterium	Beschreibung	Beispiel oder Formel
Roboter im Video	Die Anzahl der Roboter, die von dem Autor zweifelsfrei erkannt werden konnten. Dadurch werden auch teilweise sichtbare Roboter durch Kontext als Roboter erkannt.	

Verclustert	Der Detektor trifft die Annahme, dass in jeder Region ein Roboter betrachtet wird. In diese Kategorie zählen alle Roboter hinein, die sich in einer Region mit mindestens einem Roboter befinden. Die Detektion von solchen Robotern ist äußerst unwahrscheinlich.	
Nicht erfassbar	Roboter die insofern nicht vollständig im Bild sind, dass deren Detektion über die Trikotfarbe ausgeschlossen ist.	
Möglich	Der Anteil der Roboter in den Videos, die weder verclustert noch nicht erfassbar sind, aber nicht erfasst wurden.	Roboter im Video - (Verclustert + Nicht erfassbar)
Detektionen	Alle Ergebnisse des Robotererkenners.	
Roboter	Die Anzahl der durch Detektion gefundenen Roboter.	Idealfall ist bei der Detektion zu sehen.
False-Positives	Eine falsche Detektion, die ein anderes Objekt als einen Roboter markiert.	
Schiedsrichter	Fehldetektion die durch Schiedsrichter verursacht wurden.	

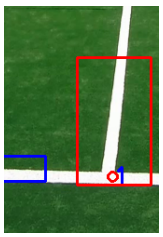
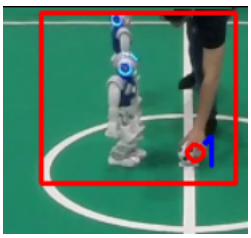
Andere	Fehldetektionen, die von etwas anderem als Schiedsrichtern ausgehen, wie Feldausschnitte und Teammitglieder.	
Vermeidbar	Die Anzahl der durch die Anbindung an den *GameController* vermeidbaren Fehler, da in bestimmten Spielphasen (Initial, Set und Finish) sehr wahrscheinlich viele Menschen über das Feld laufen.	
Erkennungsrate	Wie viel Prozent der menschlichen Wahrnehmung wird von der initialen Spielerdetektion erreicht?	$\frac{\text{Roboter}}{\text{Roboter im Video}} \cdot 100$
Fehleranteil	Wie viel Prozent der Detektionen sind False-Positives?	$\frac{\text{False-Positives}}{\text{Detektionen}} \cdot 100$
Vermeidbarer Anteil	Der Anteil der durch *GameController* Anbindung vermeidbaren Fehldetektionen.	$\frac{\text{Vermeidbar}}{\text{False-Positives}} \cdot 100$
Erwartung erfüllt	Der Anteil der Roboterdetektionen von den Systemmöglichen.	$\frac{\text{Roboter}}{\text{Möglich}} \cdot 100$

Tabelle 10.2 Übersicht der erfassten Werte.

10.3.1 1. Video: Indoor 1

Manuell erfasste Eigenschaften des Videos:

	Roboter im Video	Verclustert	Nicht erfassbar	Möglich
Anzahl	1780	1004	306	470

Auswertung der automatischen Analyse:

	Detektionen	Roboter	False-Positives	Schiedsrichter	Andere	Vermeidbar
Anzahl	194	168	24	24	0	5

Statistische Auswertung:

	Erkennungsrate	Fehleranteil	Vermeidbarer Anteil	Erwartung erfüllt
%	9,44	12,37	20,83	35,74

10.3.2 2. Video: Indoor 2

Manuell erfasste Eigenschaften des Videos:

	Roboter im Video	Verclustert	Nicht erfassbar	Möglich
Anzahl	1679	943	196	540

Auswertung der automatischen Analyse:

	Detektionen	Roboter	False-Positives	Schiedsrichter	Andere	Vermeidbar
Anzahl	134	111	24	24	0	13

Statistische Auswertung:

	Erkennungsrate	Fehleranteil	Vermeidbarer Anteil	Erwartung erfüllt
%	6,61	17,91	54,17	20,56

10.3.3 3. Video: Outdoor

Manuell erfasste Eigenschaften des Videos:

	Roboter im Video	Verclustert	Nicht erfassbar	Möglich
Anzahl	1404	659	272	473

Auswertung der automatischen Analyse:

	Detektionen	Roboter	False-Positives	Schiedsrichter	Andere	Vermeidbar
Anzahl	148	104	43	12	31	21

Statistische Auswertung:

	Erkennungsrate	Fehleranteil	Vermeidbarer Anteil	Erwartung erfüllt
%	7,41	29,05	48,84	21,99

10.3.4 Gesamt

Manuell erfasste Eigenschaften des Videos:

	Roboter im Video	Verclustert	Nicht erfassbar	Möglich
Anzahl	4863	2606	774	1483

Auswertung der automatischen Analyse:

	Detektionen	Roboter	False-Positives	Schiedsrichter	Andere	Vermeidbar
Anzahl	476	383	91	60	31	39

Statistische Auswertung:

	Erkennungsrate	Fehleranteil	Vermeidbarer Anteil	Erwartung erfüllt
%	7,88	19,12	42,86	25,83

Alle Detektionen sind als Punktwolken in Abb. 10.9 abgebildet. Über die ermittelten Punkte und die Kalibrierungen ließ sich die dreidimensionale Differenz zwischen Detektion und Kamera berechnen und darüber eine Aussage über die Detektionsreichweite treffen.

- Minimale Distanz: 1770.1098028944937 mm ≈ 1.78 m
- Maximale Distanz: 8017.537218335816 mm ≈ 8.02 m
- Durchschnittliche Distanz: 3973.906251237521 mm ≈ 3.97 m

Die durchschnittliche Distanz wurde in Abb. 10.9 als Radius für einen Kreis um die jeweilige Kameraposition verwendet und zeigt deutlich, dass die meisten Detektionen innerhalb dieses Radius liegen. Dabei sollte man beachten, dass hier ein Wert, der im dreidimensionalen Raum ermittelt wurde, auf den zweidimensionalen Raum angewandt wurde.

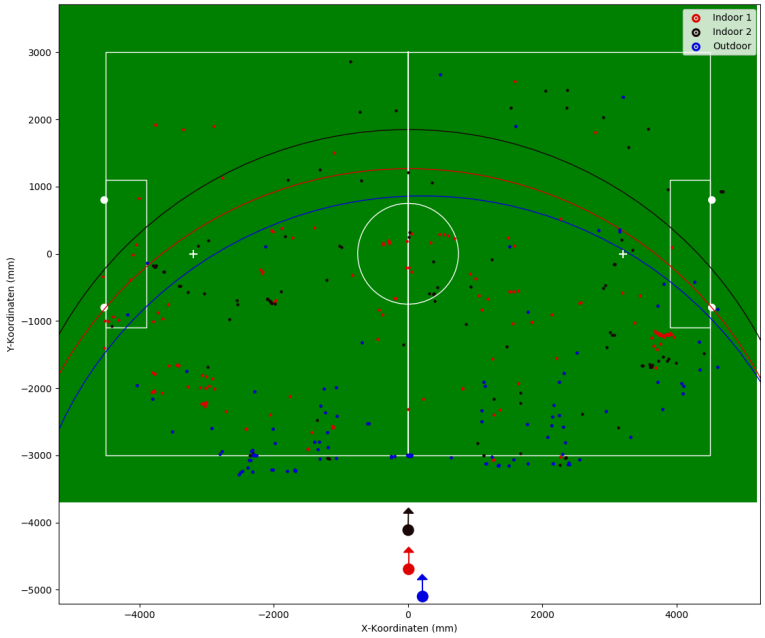


Abbildung 10.9 Das Punktwolken-Diagramm über alle Detektionen innerhalb der Evaluation der initialen Roboterdetektion. Die zugehörigen Kamerapositionen mit der mittleren Detektionsreichweite als Kreis werden der Legende entsprechend dargestellt.

10.4 Evaluation des Groundtruth-Trackings

Um die gesamtheitliche Erfassung von Robotern mit dem Tracking zu testen, soll ein gelaufener Weg eines Roboters nachvollzogen werden. Dabei werden zwei Quellen verglichen:

1. Gelaufener Weg aus den Log-Dateien des *TeamCommunicationMonitor*s: Dies ist eine Liste von Punkten, die der Selbstlokalisierung des Roboters entsprechen und deswegen auch fehlerhaft sein können.
2. Ermittelter Weg aus dem hier vorgestellten System: Diese sollten der Lokalisierung nahe kommen.

Für dieses Experiment lief der Roboter auf dem Spielfeld im Projektraum von einem Strafstoßpunkt zum anderen (von links nach rechts) auf einer möglichst geraden Linie. Der Aufbau ist in Abb. 10.10 zu sehen.

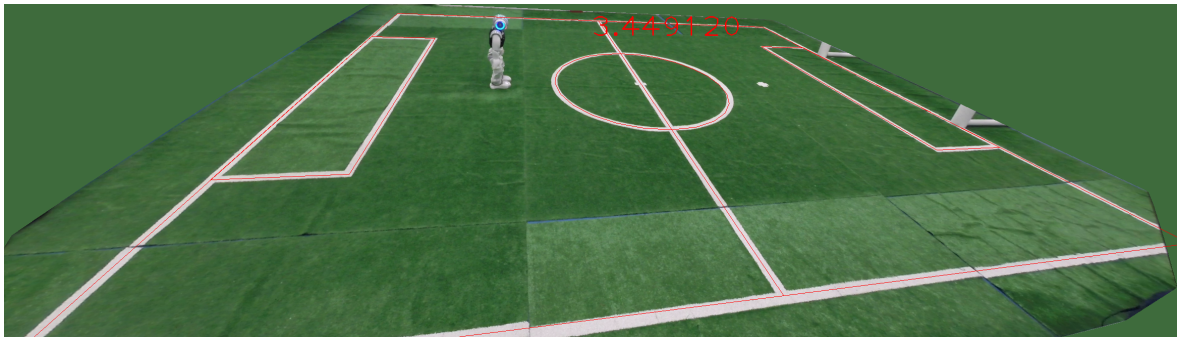
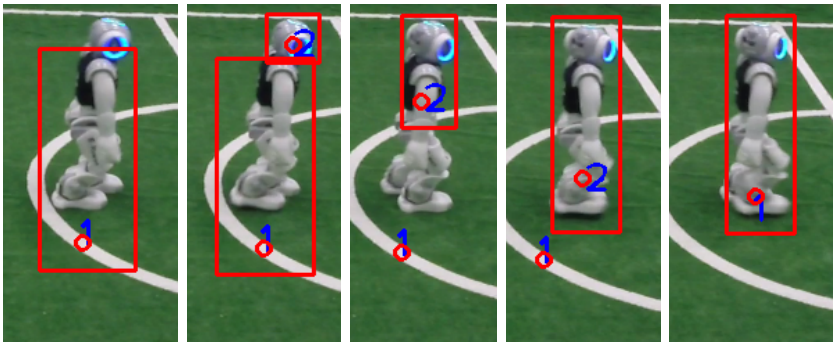


Abbildung 10.10 Der Aufbau des Groundtruth-Experiments. Der Roboter soll vom linken Strafstoßpunkt zum rechten laufen.

Idealerweise sollte der Roboter mit einer initial korrekten Lokalisierung eine perfekte gerade Linie von Strafstoßpunkt zu Strafstoßpunkt laufen, was in der Realität nicht der Fall ist. Daher wird in der vollständigen Visualisierung der Trajektorien in Abb. 10.11 darauf verzichtet, den idealen Weg darzustellen und nur die erhaltenen Trajektorien verglichen.
Wie in Abb. 10.11 zu sehen ist, wurden von dem hier vorgestellten System zwei Trajektorien ermittelt. Die zweite Trajektorie ist dem Phänomen des zu kleinen Region of Interest geschuldet, welcher im Laufe des Trackings berechnet wird. Dadurch entwickelt sich eine Detektion von Ober- und Unterkörper, wobei im späteren Verlauf der Unterkörper verworfen wurde und mit der Oberkörperschätzung fusioniert wurde. Dieses Verhalten wurde als Bilderreihe in Abb. 10.12 extrahiert und lässt sich auch aus den Trajektorien ablesen. Dieses Phänomen ist der ROI-Berechnung durch die Funktion *fitEllipse* geschuldet, bei der der Abschnitt der Linie dazu führt, dass die Ellipse keine gute Beschreibung des Roboters im Bild mehr ist. Dennoch ist an Abb. 10.12 gut zu sehen, dass die Software im Laufe der Zeit auch eigene Fehler ausgleichen kann. Dieses Verhalten lässt sich in den Daten gut lokalisieren und korrigieren, wodurch sich die Trajektorien wie in Abb. 10.13 dargestellt ergeben.
Abb. 10.13 zeigt, dass beide Trajektorien eine akzeptable Route darstellen. Wie in anderen



Abbildung 10.11 Visualisierung aller ermittelten Trajektorien für den Groundtruth-Versuch.



(a) (b) (c) (d) (e)

Abbildung 10.12 Verlauf des Trackingproblems bei dem Test der Groundtruth. In (a) wird zum ersten Mal der Region of Interest zu klein geschätzt, wodurch sich in (b) zwei Detektionen ergeben. In (c) breitet sich die Region des zweiten Trackingbereiches aus, während die erste verworfen wurde. (d) zeigt eine weiterhin wachsende Region der zweiten Detektion, die sich der ersten Schätzung nähert und in (e) letztendlich eine Fusion der beiden Schätzungen bewirkt.

Versuchen auch ersichtlich wird, besteht hier ein Verzug der Schätzung durch die Linien, der im Vergleich zu der Lokalisierung den Fehler der zu weit entfernten Projektion korrigiert. Für weitere Analysen ist in Abb. 10.14 eine y-Koordinatenbasierte Fehlervisualisierung zum idealen Laufweg zu sehen. Die unterschiedliche Anzahl der Punkte in den Visualisierungen

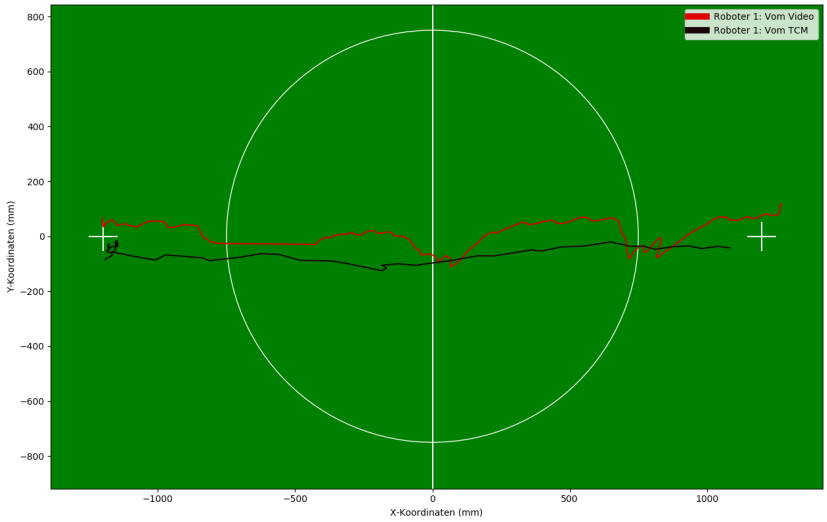


Abbildung 10.13 Visualisierung der Trajektorien ohne die durch den Fehler verursachten Ausreißer.

liegt darin begründet, dass der Roboter jede Sekunde nur 5 Nachrichten versendet, während das Video für jede Sekunde 30 Bilder enthält.

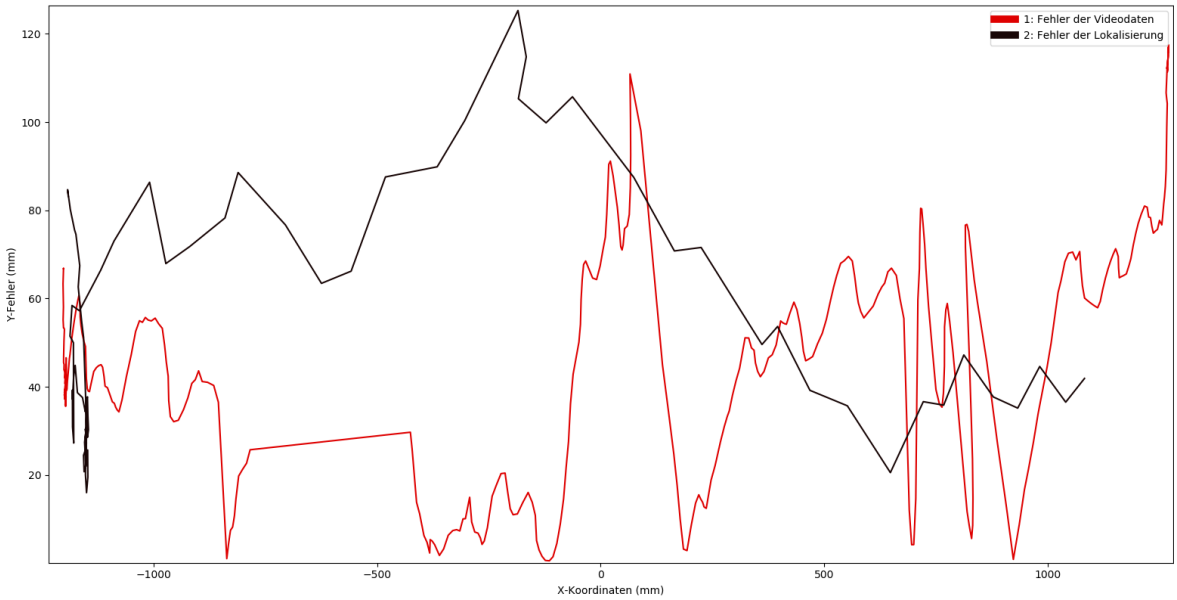


Abbildung 10.14 Visualisierung des Fehlers in der Y-Koordinate zum idealen Laufweg.

Wenn man über alle Punkte der Trajektorien den absoluten durchschnittlichen Unterschied in der y-Koordinate zum idealen Laufweg ermittelt, so sind die Werte wie folgt:

- Videodaten: 47.32295123627288 mm
- Lokalisierung: 50.649343895348835 mm

10.5 Evaluation der Farbkalibrierung

Eine grundlegende Komponente des Systems ist die Farbkalibrierung über ein angelerntes *Gaussian Mixture Model*. Wie stabil dieses Verfahren ist, wird evaluiert, indem ein Aufbau mit zwei festen Roboterpositionen auf den Strafstoßpunkten des Projektraumfeldes benutzt wird. In dem Video bleiben diese Roboter fest auf diesen Positionen, während die Lichtbedingungen stark durch die Raumsteuerung (Bewegung der Rollos, Ein- und Ausschalten der einzelnen Lichter) und einer Taschenlampe verändert werden. Beispiele daraus sind in Abb. 10.15 zu sehen.



(a) Ausgeschaltetes Licht mit Bewegung in einem Rollo.

(b) Hochfahren der Rollos bei ausgeschaltetem Licht und angestrahltem *NAO*.

(c) Eingeschaltetes Licht und Rollos, die am Herunterfahren sind.

(d) Wie (c) mit Anstrahlen der *GoPro* mit der Taschenlampe.

Abbildung 10.15 Beispielbilder aus dem Evaluationsvideo für die Farbkalibrierung.

Das Video wird in den folgenden Unterabschnitten auf zwei verschiedene Weisen analysiert. In dem ersten Abschnitt wird die initiale Farbkalibrierung, wie in Kapitel 6 vorgestellt, verwendet, während bei dem zweiten eine Analyse verwendet wird, die auf jedem Bild die Farbverteilung neu berechnet. Dabei werden in der zweiten Variante Schwarz und Weiß nicht über Referenzwerte, sondern über den minimalen und maximalen V-Wert im HSV-Raum bestimmt.

Zum Vergleich werden alle generierten Positionsdaten der Schätzungen statistisch ausgewertet. Dazu werden alle Positionen, die weiter als 20 cm von den Strafstoßpunkten entfernt sind, als unzulässige Detektion betrachtet. Ermittelt wird bei jedem Versuch die Gesamtanzahl der Schätzungen, die Anzahl der validen Schätzungen, der Anteil der validen Schätzungen und der

Anteil der validen Schätzungen über alle 1786 Bilder des Videos (in dem Sinne, dass für jedes Bild zwei valide Schätzungen existieren). Auch bei dieser Auswertung werden Prozentwerte auf bis zu zwei Nachkommastellen gerundet.

10.5.1 Konstante Farbspezifikation

Die Analyse, die nur die initiale Farbkalibrierung verwendete, produzierte Ergebnisse wie in Abb. 10.16 dargestellt. Daran ist zu erkennen, dass Roboter deutlich besser erkannt werden, wenn die Beleuchtung den ursprünglichen Grünton im Feld reproduziert, aber dass auch bei anderen Beleuchtungsbedingungen Detektionen möglich sind.

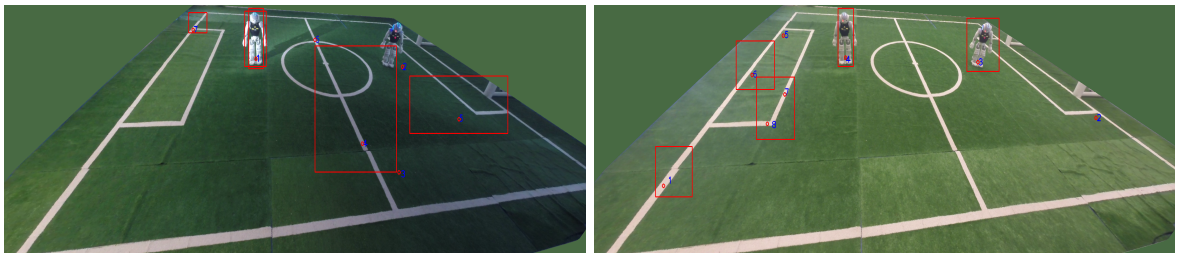


Abbildung 10.16 Beispielbilder aus der Analyse mit konstanter Farbspezifikation.

Die statistischen Werte für diese Analyse sind:

Anzahl der Schätzungen	11449
Anzahl der validen Schätzungen	3297
Anteil der validen Schätzungen an allen Schätzungen	28.8%
Anteil der validen Schätzungen zur Bilderzahl	92.3%

10.5.2 Adaptive Farbspezifikation

Die Analyse, die eine adaptive Farbkalibrierung verwendete, produzierte Ergebnisse wie in Abb. 10.17 dargestellt. In den Bildern ist an den eingezeichneten Regionen, die außerhalb des Feldes liegen, zu sehen, dass sich die initiale Grünschätzung stark verändert hat. Es sind dadurch immer noch Detektionen möglich, aber es sind auch wie bei der konstanten Variante False-Positives entstanden.

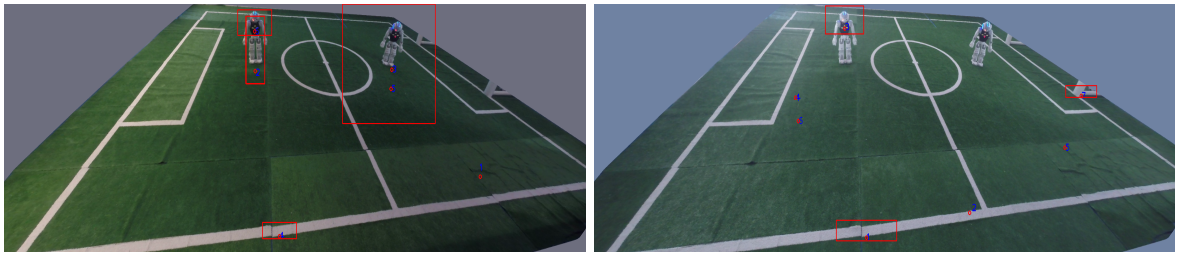


Abbildung 10.17 Beispielbilder aus der Analyse mit adaptiver Farbspezifikation.

Die statistischen Werte für diese Analyse sind:

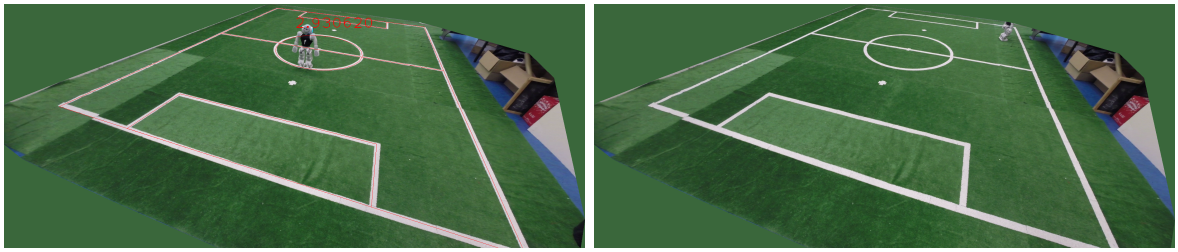
Anzahl der Schätzungen	8047
Anzahl der validen Schätzungen	2318
Anteil der validen Schätzungen an allen Schätzungen	28.81%
Anteil der validen Schätzungen zur Bilderzahl	64.89%

10.6 Evaluation der Blickwinkelunabhängigkeit

Durch den Detektionsalgorithmus werden nur indirekt Annahmen über die Kameraposition getroffen und wie eine optimale Perspektive auszusehen hat. Ohne konkrete Einschränkungen sollte jedes Video, welches vom Feldrand aufgenommen wurde, analysierbar sein. Um dies zu bestätigen, wurden vier Videos von unterschiedlichen Feldrandpositionen aufgenommen, während ein Roboter immer nur geradeaus läuft. Durch den maschinellen Verschleiß lief der Roboter gleichmäßig eine Kurve. Dieses Ergebnis sollte in den Daten aller Videos sichtbar sein. Die Resultate sind in den Unterabschnitten separat mit dem ersten und letzten Bild der Analyse und den resultierenden Referenzdaten aufgelistet.

10.6.1 Erster Aufbau

Der erste Versuch der Blickwinkelunabhängigkeit nutzt eine Kameraposition, die von der kürzeren Seite des Feldes das Spielgeschehen erfasst. Dieses Szenario ist recht häufig, da die seitlichen Positionen häufig durch das Schiedsrichtersetup oder durch Reportagedienste besetzt werden. Der Aufbau und die generierten Daten sind in Abb. 10.18 abgebildet. Wie in (a) und (b) zu sehen ist, wurden die Regionen außerhalb des Feldes nicht vollständig erfasst.



(a) Anfangsbild (b) Endbild

Abbildung 10.18 Der Aufbau des ersten Versuchs der Blickwinkelunabhängigkeit.

Dies liegt an der Nähe von Braun zu Grün im HSV-Farbraum in Kombination mit der Regionsidentifikation über die konvexe Hülle, wodurch die braunen Pakete ihr Umfeld sichtbar machen. Da die Pakete und ihre Nachbarschaft sich deutlich außerhalb des Feldes befinden, sind in ihnen keine False-Positives vorhanden, wie in Abb. 10.19 ersichtlich. Die initialen Schwierigkeiten bei der Erfassung des Roboters basieren auf einem Fitting-Problem der Ellipse, die für kurze Zeit ein zu kleines ROI berechnet hat. Ansonsten lässt sich der Weg des

Roboters deutlich nachvollziehen, wobei die einzelnen Schwankungen auf die Schwerpunktverschiebung beim Laufen geschoben werden können. Die von der Kalibrierung berechnete Kameraposition ist zu weit auf dem Feld, aber liegt in der richtigen Richtung und hat eine plausible Ausrichtung.

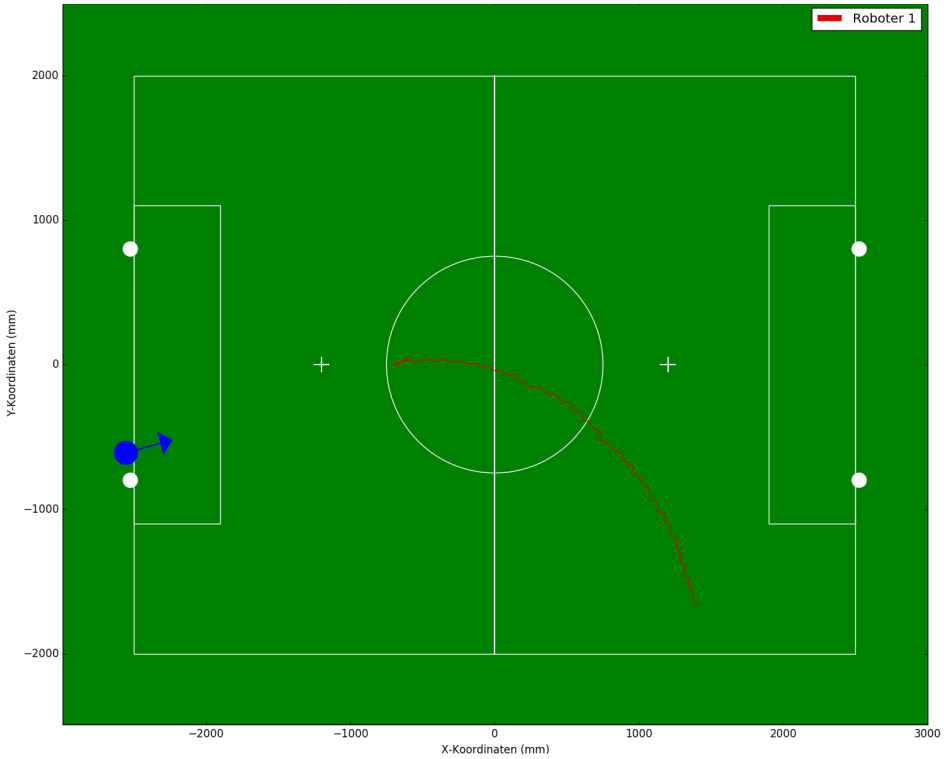
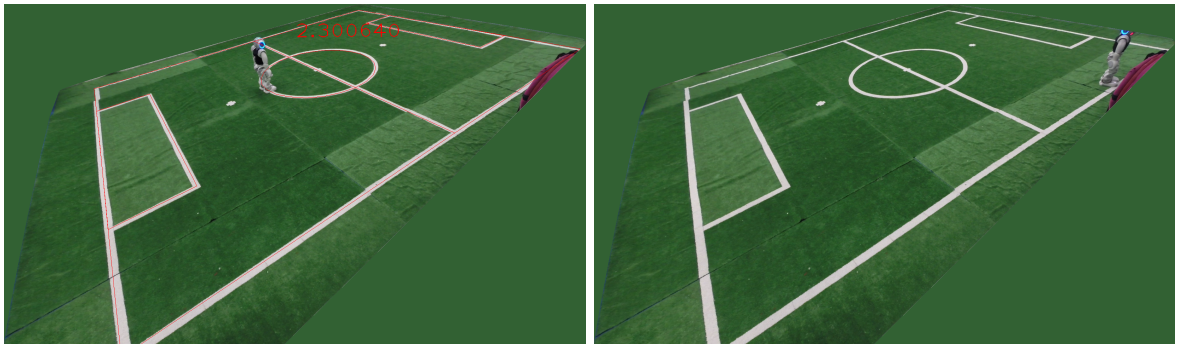


Abbildung 10.19 Der generierte Pfad des ersten Versuchs der Blickwinkelunabhängigkeit.

10.6.2 Zweiter Aufbau

Beim Aufbau des zweiten Versuchs in Abb. 10.20 (a) kann man sehen, dass eine rosa Tasche in das sichtbare Feld hineinragt. Durch die Bestimmung der Regionen, die außerhalb des Feldes liegen, mittels einer konvexen Hülle, wird die Tasche für die weitere Analyse sichtbar. Besonders ungünstig dabei sind die schwarzen Streifen der Tasche, die, wie in Abb. 10.21 zu sehen ist, False-Positives erzeugen, welche sich durch die ROI-Berechnung im Tracking verschieben. Dennoch ist die Route des Roboters klar zu identifizieren, auch wenn sie zum Ende hin durch die Nähe zur Feldlinie abdriftet. Die Kameraposition ist diesmal nicht zu nah am Feld geschätzt, sondern eher zu weit entfernt, aber die Ausrichtung ist korrekt.



(a) Anfangsbild (b) Endbild

Abbildung 10.20 Der Aufbau des zweiten Versuchs der Blickwinkelunabhängigkeit.

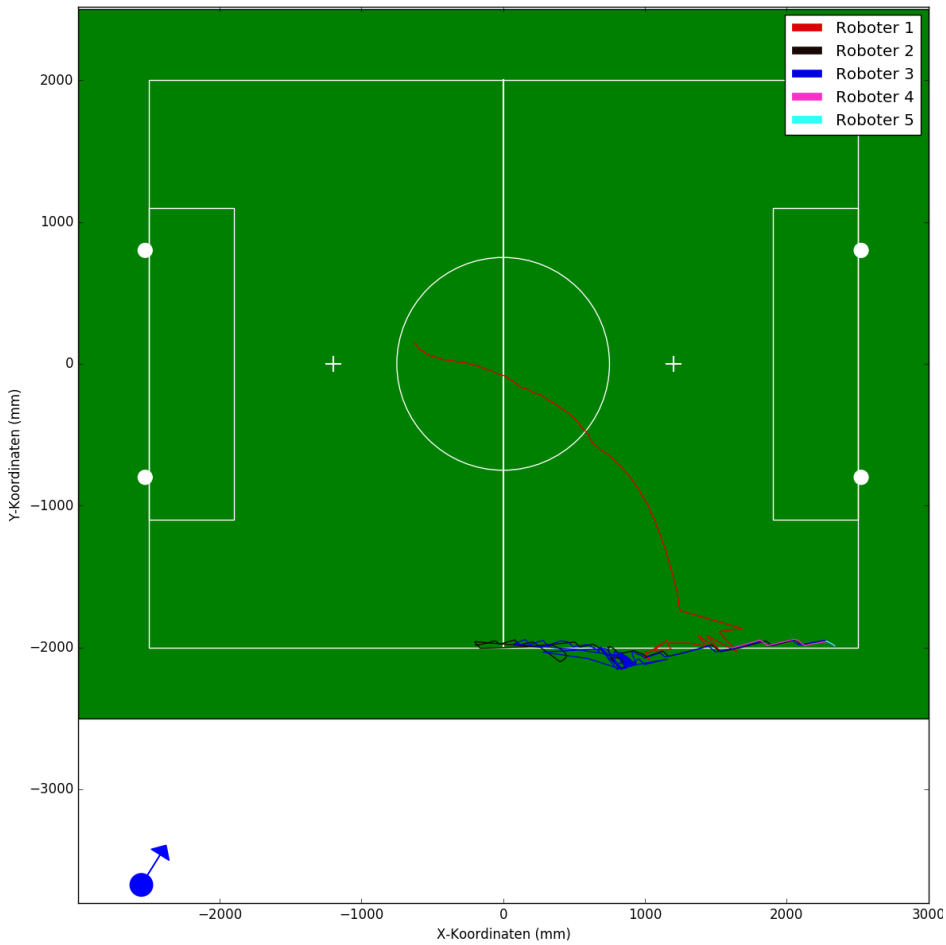
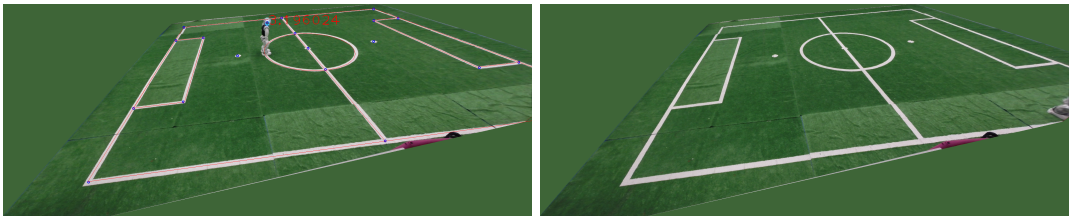


Abbildung 10.21 Der generierte Pfad des zweiten Versuchs der Blickwinkelunabhängigkeit.

10.6.3 Dritter Aufbau

In Kapitel 10.22 (a) ist wieder ein Ausschnitt der rosa Tasche zu sehen, der auch einen Schwarzanteil besitzt. Dadurch entstehen analog zu Versuch 2 False-Positives.
In Abb. 10.23 ist ein deutlicher Ausreißer beim Verlassen des Mittelkreises zu sehen. Dieser lässt sich durch eine Verfälschung der Position durch den Linienanteil im ROI erklären,



(a) Anfangsbild (b) Endbild

Abbildung 10.22 Der Aufbau des dritten Versuchs der Blickwinkelunabhängigkeit.

weshalb nach einem gewissen Abstand zum Mittelkreis der Roboter gleichmäßiger erfasst wird. Die Kameraposition wird auf dem Feld geschätzt, aber besitzt auch hier eine korrekte Ausrichtung.

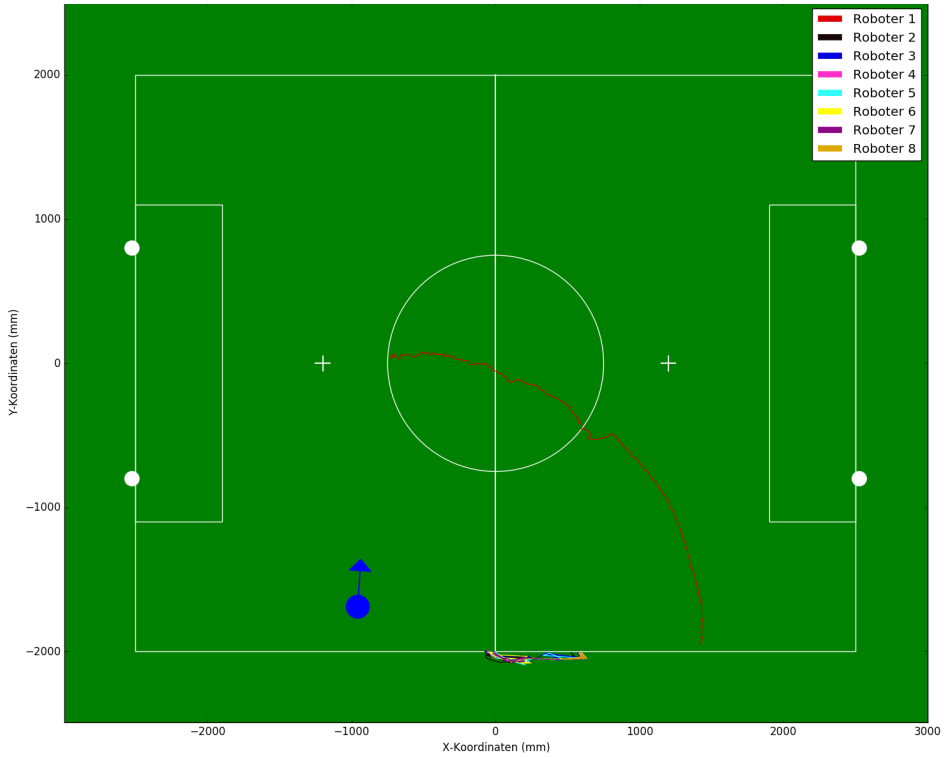
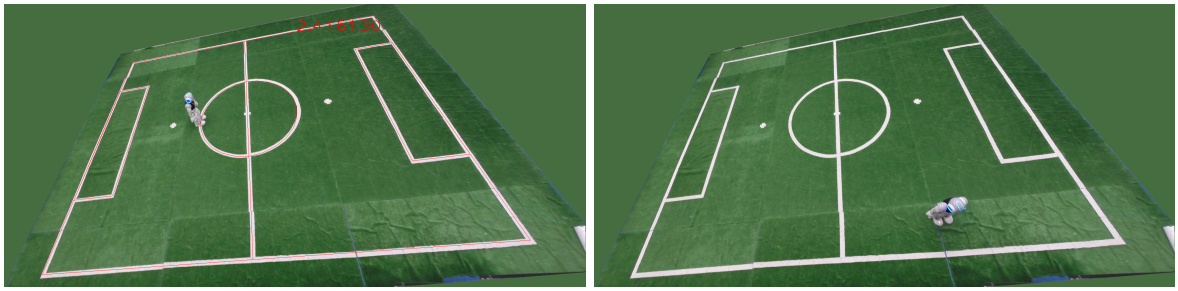


Abbildung 10.23 Der generierte Pfad des dritten Versuchs der Blickwinkelunabhängigkeit.

10.6.4 Vierter Aufbau

Der vierte Versuch fand von einer erhöhten Kameraposition aus statt, wie in Abb. 10.24 (a) und (b) zu sehen ist. Dadurch ist eine deutlich andere Perspektive gegeben, die die Berechnung von ROIs verkompliziert, aber dafür auch die Feldregion besser begrenzt.
Die resultierende Trajektorie in Abb. 10.25 zeigt anfängliche Ausreißer, die mit den Linien des Mittelkreises zusammenhängen. Im späteren Verlauf sind größere Schwankungen zu sehen, die sich auch mit dem Schwerpunktwechsel erklären lassen. Die Kameraposition wird im Vergleich



(a) Anfangsbild (b) Endbild

Abbildung 10.24 Der Aufbau des vierten Versuchs der Blickwinkelunabhängigkeit.

zu Versuch 3 noch weiter in das Feld geschätzt, die, wäre sie korrekt, kein vollständiges Tracking ermöglichen würde. Dennoch ist die Ausrichtung plausibel.

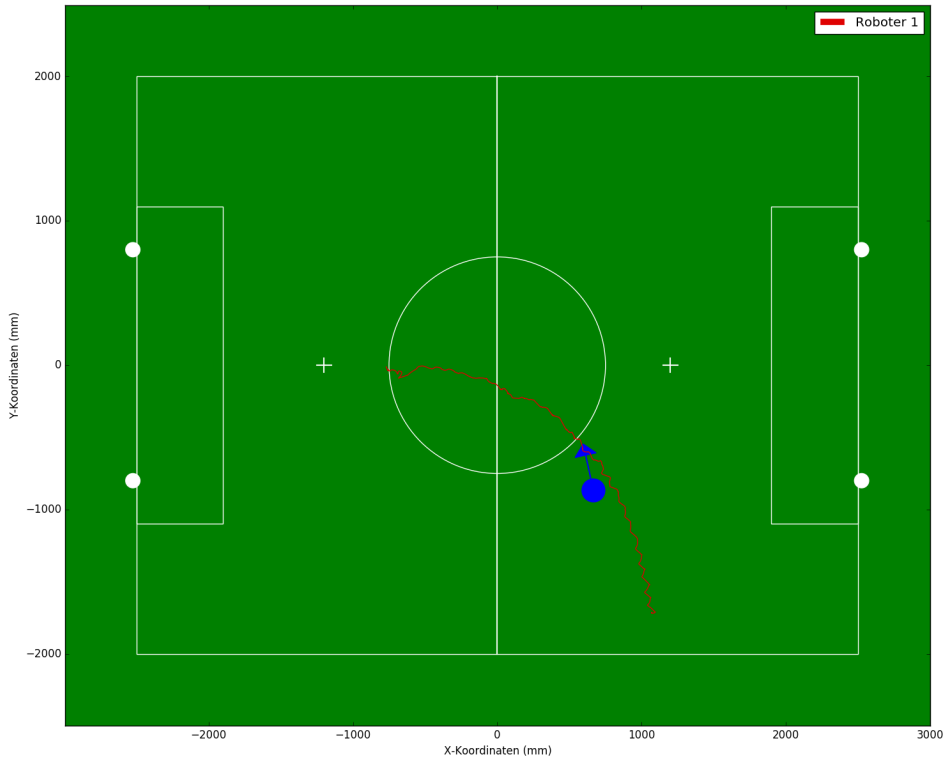


Abbildung 10.25 Der generierte Pfad des vierten Versuchs der Blickwinkelunabhängigkeit.

10.6.5 Gesamtvergleich

Alle zuvor ausgeführten Versuche konnten die Trajektorie des Roboters erfassen. Dabei gab es bei zwei Versuchen False-Positives, die durch einen störenden Gegenstand verursacht wurden. In den CSV-Dateien lassen sich durch die generierten Visualisierungen die tatsächlichen Roboter leicht identifizieren und in einer eigenen CSV-Datei extrahieren. Wenn man diese visualisiert, erhält man das Bild aus Abb. 10.26. Das Ergebnis verdeutlicht, dass trotz der un-

terschiedlichen Perspektiven ähnliche Daten bei gleicher Versuchsausführung erzielt werden. Die von der Kamerakalibrierung errechneten Kamerapositionen und -ausrichtung lassen eine klare Unterscheidung der verwendeten Perspektive zu, soweit es auf einer zweidimensionalen Darstellung möglich ist. Dennoch ist es interessant, dass nur in einem Video die Kamera tendenziell zu weit vom Feld geschätzt wird und in allen anderen zu nahe.

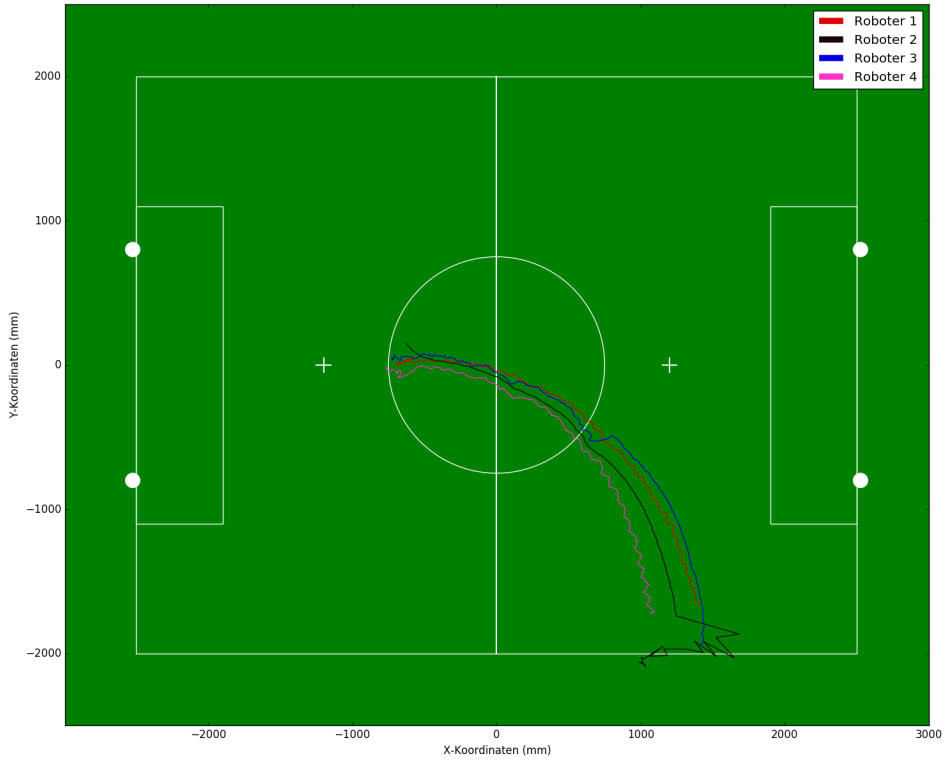


Abbildung 10.26 Zum Vergleich eingezeichnete Trajektorien des Roboters über alle vier Versuche.

10.7 Evaluation der Auswirkung von Verdeckung

Um die Auswirkungen von Verdeckungen abzuschätzen, sollen sich zwei Roboter auf dem Projektraumfeld begegnen. Dazu wird in den beiden Versuchen ein Roboter statisch auf einem Punkt stehen, während ein zweiter Roboter von dem linken Strafstoßpunkt zum rechten läuft. Der Hindernisroboter wurde immer auf gleicher X-Koordinatenhöhe gesetzt und 90 cm von der Mitte entfernt aufgestellt. Die Hindernispositionen der beiden Versuche waren:

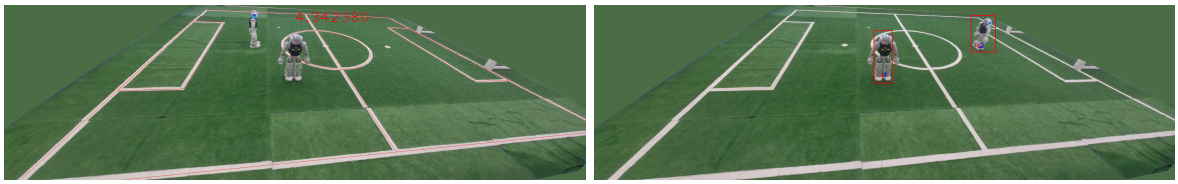
$$\text{Erster Versuch:} \qquad P_1 = \begin{pmatrix} -75 \text{ cm} \\ -90 \text{ cm} \end{pmatrix} \tag{10.6}$$

$$\text{Zweiter Versuch:} \qquad P_2 = \begin{pmatrix} -75 \text{ cm} \\ 90 \text{ cm} \end{pmatrix} \tag{10.7}$$

Die generierten Daten werden mit den Trajektorien der Lokalisierung verglichen, die aus den Aufzeichnungen des *TeamCommunicationMonitors* (TCM) extrahiert wurden.

10.7.1 Versuch 1

Der Aufbau für diesen Versuch ist in Abb. 10.27 mit der Anfangssituation und der Endsituation zu sehen.



(a) Anfangsbild (b) Endbild

Abbildung 10.27 Der erste Versuch der Verdeckungsevaluation.

In diesem Durchgang hatte der Roboter keine Probleme beim Laufen und produzierte, als er dem stehenden Roboter zu nahe kam, eine Verdeckung wie in Abb. 10.28 dargestellt. Dies hatte zur Folge, dass die ursprüngliche Schätzung und Messung des laufenden Roboters zu dem Stehenden konvergierten.

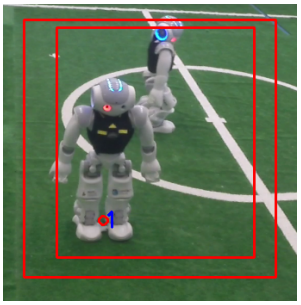


Abbildung 10.28 Die entstandene Verdeckung im ersten Versuch.

Während der laufende Roboter die Deckung verließ, gab es mehrere neue Schätzungen für seine Position, die mit seinem sichtbaren Kopf begannen und gelegentlich auf den stehenden Roboter zurückfielen, bis der Laufende deutlich zu unterscheiden war. Dabei konnte es auch geschehen, dass die Trackingregion zu groß wurde und ein Fußpunkt in einer Linie gefunden wurde. Das Geschehen lässt sich an Abb. 10.29 ablesen.
Analog zum Groundtruthtest kann man auch hier leicht die erhaltenen Werte modifizieren, damit man nur die gewünschten Trajektorien erhält. Das Ergebnis dieser Selektion ist in Abb. 10.30 zu sehen, bei denen die Ausreißer des stehenden Roboters, die durch Fusion der Schätzungen entstanden, sehr schwer zu identifizieren sind und daher noch in der Visualisierung enthalten sind. Ebenfalls dargestellt ist die vom Roboter wahrgenommene Laufstrecke.

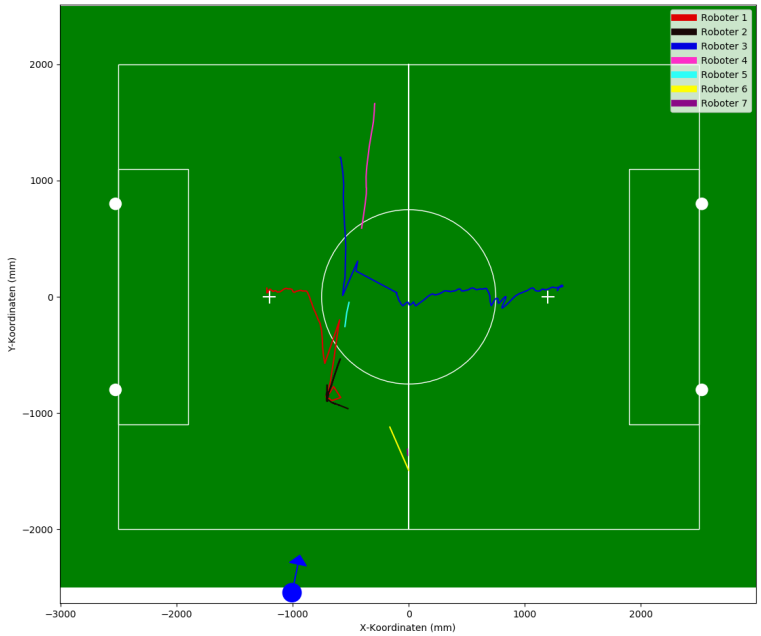


Abbildung 10.29 Die generierten Daten aus dem ersten Verdeckungsversuch.

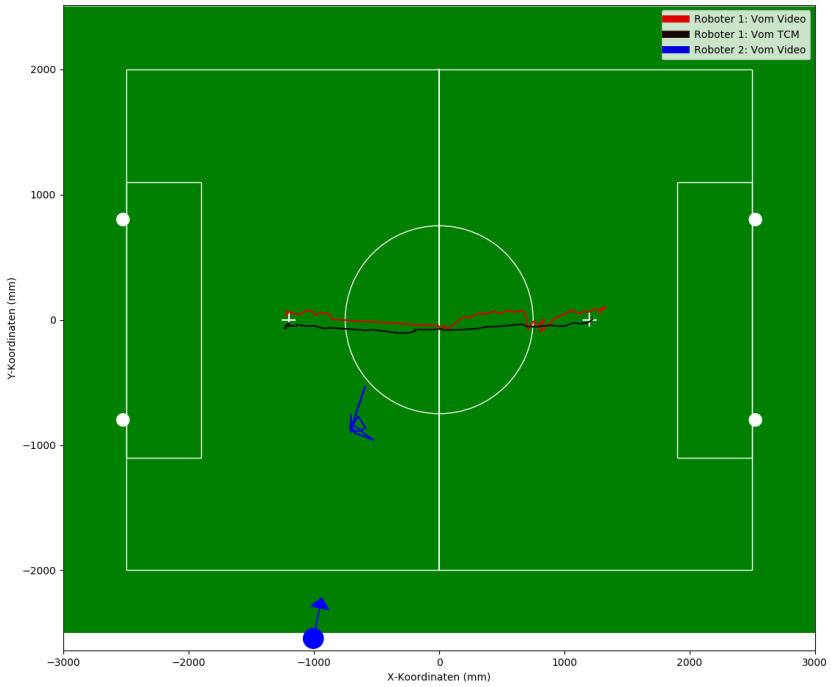
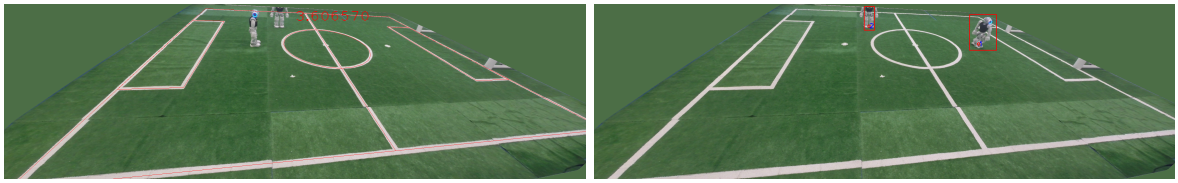


Abbildung 10.30 Visualisierung der manuell verbesserten Trajektorien der Roboter im Vergleich zu der Lokalisierung des laufenden Roboters aus dem erstem Versuch.

10.7.2 Versuch 2

Der Aufbau für den zweiten Versuch ist in Abb. 10.31 mit der Anfangssituation und der Endsituation zu sehen.
In diesem Durchgang hatte der Roboter Probleme beim Überqueren des Mittelkreises und produzierte dabei Schwankungen in den Messungen. Bei ausreichender Nähe zum stehenden



(a) Anfangsbild (b) Endbild

Abbildung 10.31 Der zweite Versuch der Verdeckungsevaluation.

Roboter kam eine Verdeckung wie in Abb. 10.32 dargestellt zustande. Dadurch konvergierten Schätzung und Messung des stehenden Roboters zu dem Laufenden, da dieser sich im Vordergrund befand.

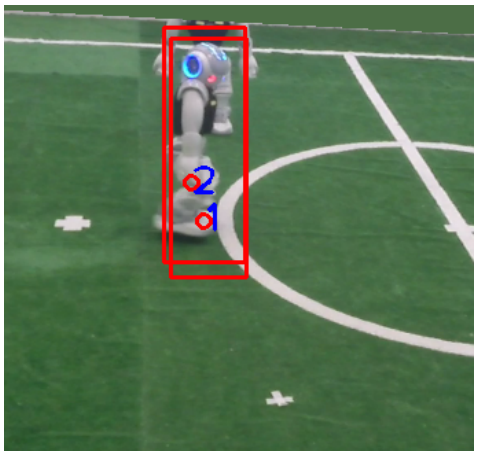


Abbildung 10.32 Die entstandene Verdeckung im zweiten Versuch.

Während des Versuchs gab es einen False-Positive im linken Torraum, während die Schätzung des stehenden Roboters das Tracking des Laufenden übernahm. Als die Trackingregion sich nach der Verdeckung auf den Laufenden anpasste, war auch die Erfassung des Stehenden möglich. Dieses Verhalten lässt sich auch an Abb. 10.33 ablesen.
Auch für diesen Versuch wurde eine manuelle Extraktion der gewünschten Trajektorien durchgeführt und mit der Lokalisierung des Roboters, wie in Abb. 10.34 dargestellt, verglichen.

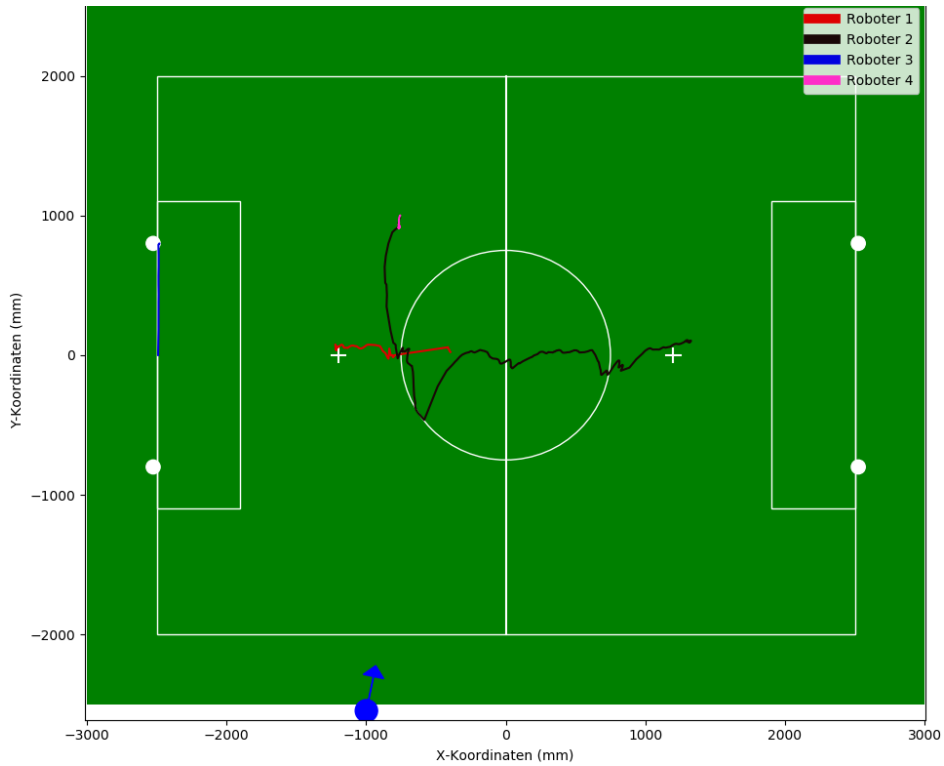


Abbildung 10.33 Die generierten Daten aus dem zweiten Verdeckungsversuch.

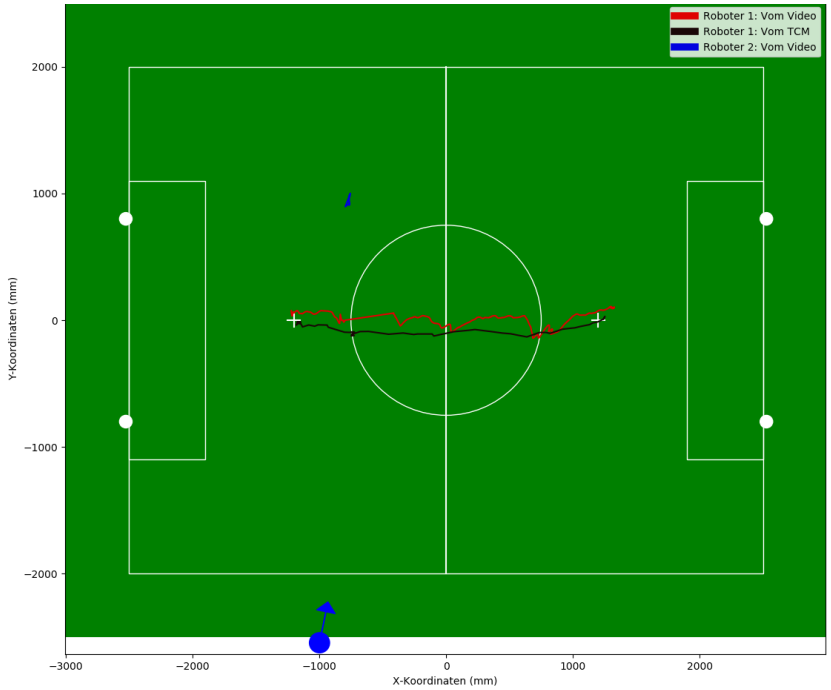


Abbildung 10.34 Visualisierung der manuell verbesserten Trajektorien der Roboter im Vergleich zu der Lokalisierung des laufenden Roboters aus dem zweiten Versuch.

10.8 Laufzeit-Betrachtungen

In diesem Abschnitt werden die Laufzeiten einiger Komponenten betrachtet, die für dieses System entscheidend sind, und einige wenige, die nur für Entwicklungszwecke interessant waren. Die Berechnung der Laufzeit wurde mit der C++-Bibliothek <chrono> im Millisekunden-Format auf einem Video der Indoor Competition des RoboCup 2016 in Leipzig mit einer Auflösung von 1920×1080, dargestellt in Abb. 10.35, über 200 Bilder in der Release-Build-Konfiguration auf einer i7-4790 CPU durchgeführt.



Abbildung 10.35 Das für die Laufzeit analysierte Video des RoboCup 2016 in Leipzig.

Die Ergebnisse sind nach Kategorien geordnet in Tab. 10.3 zu sehen. Alle Werte sind nach Möglichkeit auf bis zu zwei Nachkommastellen gerundet. Es gibt allerdings Einträge, die denselben Wert für Minimum, Maximum und Durchschnitt besitzen, was daran liegt, dass diese Routinen nur einmal im Gesamtverlauf ausgeführt wurden, wie zum Beispiel die initiale Farbkalibrierung, weshalb sich diese Einträge über drei Spalten erstrecken.

Modul/Funktion	Minimum (ms)	Maximum (ms)	Durchschnitt (ms)
Analysekette			
Kettendurchgang	140.92	2231.19	274.73
Roboterroutine	140.77	471.17	264.77
Kamerakalibrierung			
Entzerrung eines Bildes	150.44	215.52	167.25
Bild-Zu-Welt-Transformation	0.005	0.09	0.02
Welt-Zu-Bild-Transformation	0.006	0.38	0.01
Hypothesenverwaltung			
Aktualisierung über Messungen	1.27	11.24	5.85
Aussortieren	0.0	0.001	0.0004
Clustern	0.0009	0.01	0.002
Aktualisierung des Speichers	0.0006	0.01	0.002
Speichern der Daten	4.35		

Bildvorverarbeitung			
Gesamtaktualisierung	197.47	456.05	240.08
Anwendung der Vorverarbeitung	0.0003	217.36	168.22
Erstellung von Zusatzbildern	59.45	181.78	67.01
Graustufenbild	0.32	1.1	0.37
Sobel-Bild	2.61	9.83	3.2
Farbkalibrierung			
Gesamtkalibrierung	1032.47		
Erstellung der Trainingsmenge	0.16		
Anlernen des GMM	15.39		
Berechnung des Feldausschnitts	972.92		
Berechnung der Hintergrundregionen	6.03		
Segmentierung	0.18	964.52	15.46
Kalman-Filter			
Aktualisierung	0.0006	6.02	1.2
Robotererkenner			
Ganzer Durchlauf	133.32	465.71	257.45
Testen eines bereits getrackten Roboters	0.92	243.27	51.14
Kandidatenuntersuchung	0.0009	97.59	6.42
Feature-Clustering	1.44	9.51	1.84
Konturextraktion	0.03	1.37	0.18
Test auf einfache Verdeckung	0.03	0.7	0.09
Fußdetektion	0.03	2.82	0.82
Videozugriff			
Bildzugriff	4.55	27.3	7.14
Nicht für die Anwendung benötige Vorverarbeitung			
Canny-Bild	1.44	6.04	1.75
Backgroundsubtractor	8.33	72.23	9.91

Tabelle 10.3 Übersicht der Laufzeiten.

10.9 Test des Gesamtsystems

Um die Anwendbarkeit des Gesamtsystems im Kontext des RoboCups zu eruieren, wurde das Video von Abb. 10.35 über 3 Minuten analysiert. Zum Vergleich wurden die Trajektorien der Roboterlokalisierungen, die aus den ersten drei Minuten der *TeamCommunicationMonitor*-Logs extrahiert wurden, aus diesem Spiel in Abb. 10.36 eingezeichnet.

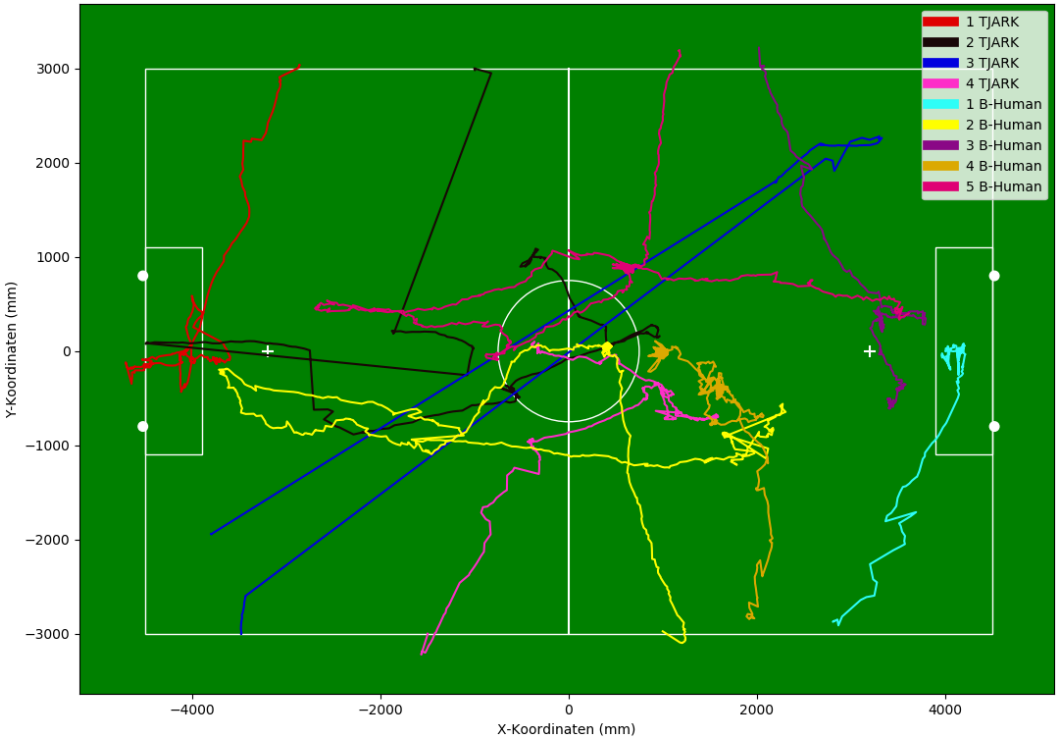


Abbildung 10.36 Die ermittelten Trajektorien aus der Teamkommunikation.

Durch eine manuelle Analyse lassen sich in Abb. 10.36 zwei Fehler im Team TJARK identifizieren, da in dem Video auf einer Seite drei Roboter einlaufen und auf der anderen nur einer. Deren dritter Roboter traf initial eine falsche Annahme und spielte anschließend an gespiegelter Position. Ebenso hatte deren Nummer 2 mehrere Probleme und sprang häufiger in der Lokalisierung hin und her. Im Vergleich dazu lässt sich aus den B-Human-Robotern sehr angenehm die Anstoßstrategie ablesen.
Die vollständig extrahierten Daten über die 10800 ausgewerteten Bilder der ersten drei Minuten sind in Abb. 10.37 zu sehen.
Abb. 10.37 zeigt, dass die Fülle der Daten deren anschließende Überprüfung nötig macht. Ebenso wären mehr Modellierung und Mechanismen nötig um kontinuierliche Trajektorien zu ermöglichen, da in dem aktuellen Stand Verdeckungen sehr schnell die Ergebnisse verfälschen. Zum Vergleich sind in Abb. 10.38 manuell selektierte und zusammengefasste Trajektorien visualisiert. Die Selektion schloss Trajektorien aus, die wesentlich zu kurz sind und verband diejenigen, die nach *TeamCommunicationMonitor* wahrscheinlich zusammen gehören.
In Abb. 10.38 wurden die manuell platzierten Roboter auch erfasst und viele Bewegungen lassen sich nachvollziehen. Jedoch ist immer noch ersichtlich, dass Kreuzungen der Pfade mit Feldlinien die Daten verfälschen. Dennoch sind einige Fragmente bei der Selektion verloren gegangen, die in Kombination mit anderen doch noch nützliche Informationen enthalten hätten. Beachtlich ist, dass die 2 von TJARK nahezu durchgängig nach der manuellen Platzierung erfasst wurde. Ebenso lässt sich auch hier in den B-Human-Robotern die Spielstrategie ablesen, bei denen nach dem Einlaufen die 1 immer der Torwart ist, während 3 und 4 die Verteidiger

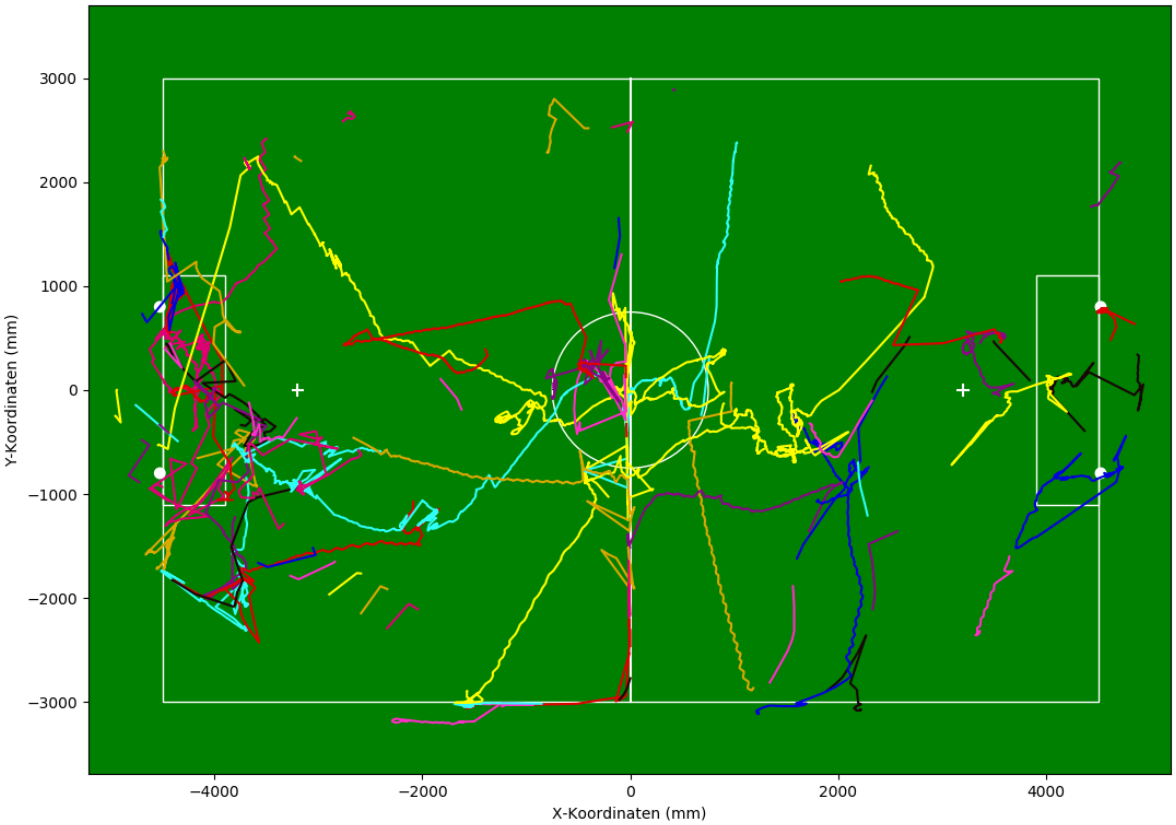


Abbildung 10.37 Alle ermittelten Trajektorien aus der Videoanalyse.

sind. Zusätzlich lässt sich ablesen, dass die 2 als Stürmer ein Tor schießt und anschließend wieder auf die eigene Spielfeldhälfte zurückkehrt. Analog ist es bei der 5, die zur Torschussunterstützung die 2 begleitet und anschließend einen Weg zur eigenen Spielfeldhälfte einschlägt. Die Anstoßsituation ist in Abb. 10.39 zu sehen.

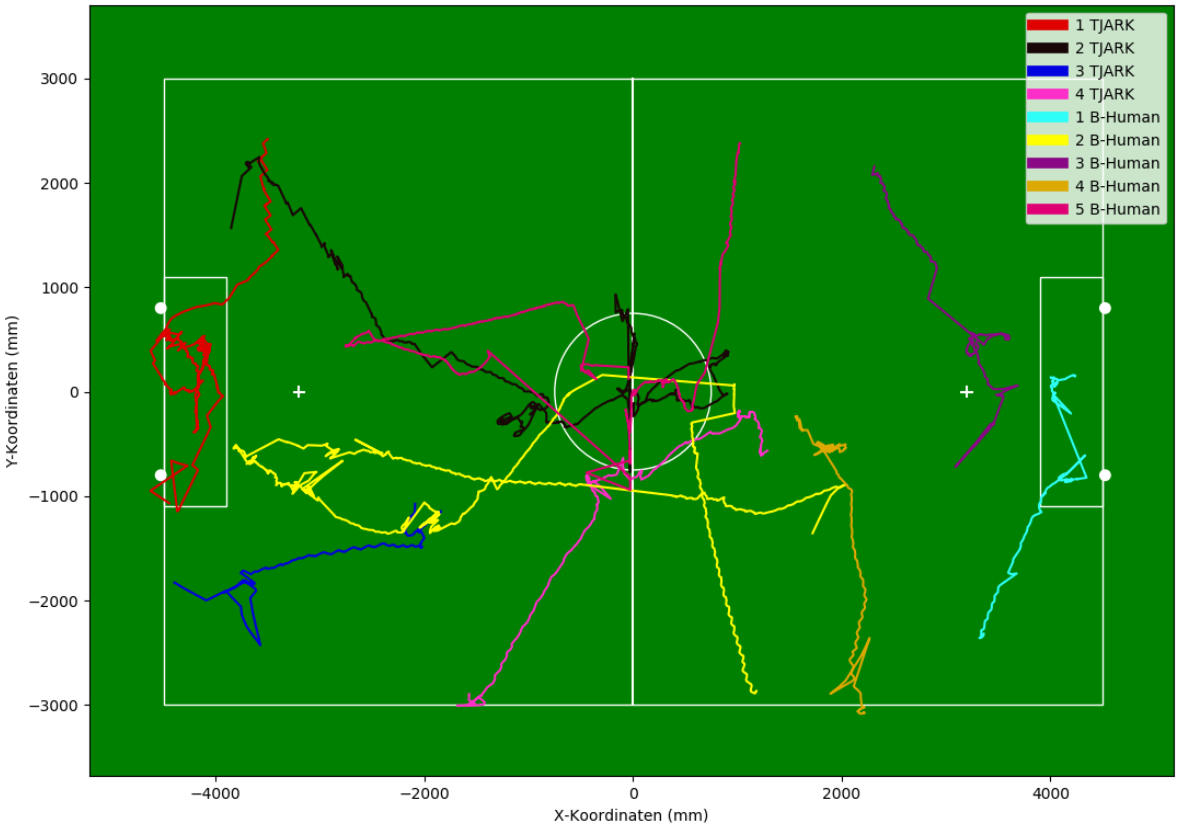


Abbildung 10.38 Visualisierung der manuell herausgearbeiteten Trajektorien aus den Daten.

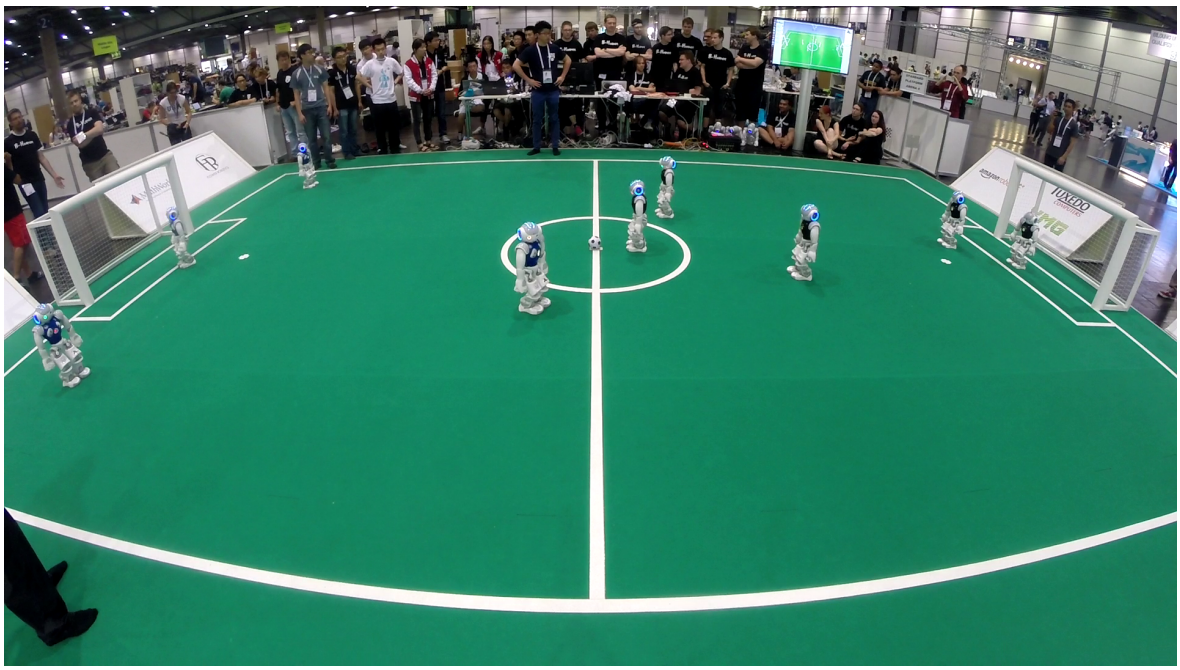


Abbildung 10.39 Die Anstoßsituation aus dem evaluierten Video.

Kapitel 11

Fazit und Ausblick

In diesem Kapitel werden die Versuche aus dem Kapitel Evaluation analysiert und die insgesamt erbrachte Leistung in dem Rahmen dieser Arbeit kritisch betrachtet. Anschließend wird ein Ausblick auf eventuelle Weiterentwicklungen in dem Themenbereich gegeben. Am Ende des Kapitels wird ein Fazit für diese Arbeit gezogen.

11.1 Auswertung

Die Evaluation in dem vorherigen Kapitel hat einige Grenzen und Möglichkeiten des Systems klar aufgezeigt. In Kapitel 10.1, dem Vorwort zur Evaluation, wurde deutlich, dass schwierige Lichtverhältnisse und ein ungleichmäßiger Farbverlauf nicht durch die initiale Farbkalibrierung kompensiert werden können. Dieses Problem könnte mit einer aktiven Kamerasteuerung im Sinne von Einstellung der Beleuchtungsdauer und weiteren Parametern kompensiert werden. Dies sollte künftig im Kontext des RoboCups im Rahmen automatischer Kalibrierungsverfahren für Farben weiter untersucht werden.

11.1.1 Der Detektionsfehler

Die Untersuchung des Detektionsfehlers in Kapitel 10.2 zeigt, dass auch liegende Roboter erkannt werden, aber auch, dass durch die farbbasierte Detektionsmethode die Reichweite recht begrenzt ist. Die Detektionsgenauigkeit wird durch die Detektion von liegenden Robotern etwas geringer, da die perspektivischen Annahmen, die getroffen wurden, nicht auf alle Fälle eines liegenden Roboters zutreffen. Jedoch liegt der allgemeine Fehler im Sinne eines Root-Mean-Squared-Errors unter 20 Zentimeter, was nicht mit kommerziellen Trackinginstrumenten konkurrieren kann, welche Marker für das Tracking verwenden. Dennoch reicht die Genauigkeit für viele Analysen im Kontext des RoboCups aus. Die Entwicklung von Khandelwal und Stone [2012] erzielte bei stehenden Robotern einen durchschnittlichen Fehler von 10 cm, wogegen hier 11,8 cm Genauigkeit erreicht wurden. Zusätzlich könnte man durch zusätzliche Modellierung versuchen, den systematischen Positionsfehler in der Detekti-

on auszugleichen, der in Abb. 10.6 angedeutet ist, der anscheinend mit der Nähe zur Kamera korreliert. Bei diesem Versuch war auch der Unterschied zwischen errechneter Kameraposition und tatsächlicher Kameraposition bemerkenswert, obwohl trotzdem annehmbare Projektionen erzielt wurden.

11.1.2 Die initiale Spielererkennung

Das Experiment für die initiale Spielererkennung in Kapitel 10.3 erzielte über das analysierte Videomaterial fast eine Detektionsrate von 8 % der menschlichen Wahrnehmung, was durchaus beachtlich ist, da der Mensch sehr viele seiner visuellen Informationen über den Kontext auswertet. Da allerdings der Algorithmus derart entworfen wurde, dass er möglichst keine False-Positives erzeugt, lässt ein allgemeiner Anteil eben dieser von nahezu 20 % die Anwendbarkeit der Robotererkennung etwas in Zweifel ziehen. Das Problem liegt in der farbbasierten Detektion begründet, die sich hier auf Grün- und Schwarz-Wahrnehmung beschränkt. Schwarz ist eine Farbe, die in Bildern häufig vorkommt, da ein Schatten eine Farbe deutlich zu Schwarz abändern kann. Dadurch lassen sich False-Positives in Feldsegmenten nahe der Linien erklären, deren dunkles Grün mit dem Weiß der Linie den Detektor in die Irre führen kann. Ähnlich ist es mit der Kleidung der Schiedsrichter, die nach Regeln [vgl. RoboCup Technical Committee, 2017, Kapitel 5.4] in eine dunkle Richtung ermutigt wird und damit viele Möglichkeiten für False-Positives bietet. Da ein Schiedsrichter ein aktiver Teil der Spiele ist, lässt sich dieser bei ungünstiger Positionierung vor der Kamera nicht immer ausschließen. Dennoch würde eine Anbindung an die W-Lan-Kommunikation der SPL eine deutliche Reduktion an False-Positives auf einen Anteil von ca. 11 % bringen, da Detektionen von Teammitgliedern bei der Vorbereitung und Schiedsrichtern in bekannten schwierigen Spielabläufen ausgeschlossen werden können. Zusätzlich hat sich eine grobe Abschätzung der effektiven Sensorreichweite des hier vorgestellten Systems ergeben, die annähernd 4 Meter erreicht und damit für die vollständige Auswertung über ein Feld der SPL nur unzureichend ist. Nach Khandelwal und Stone [2012] ist die Reichweite der Kinect nach offizieller Spezifikation auf einen Bereich von 1,2 - 3,5 Metern beschränkt, während ihre Experimente eher eine Reichweite von 0,7 - 7 Metern ermittelt haben, wobei es auch in diesem Versuch Perzepte gab, die mehr als 7 Meter entfernt waren.

11.1.3 Das Groundtruthtracking

Die Evaluation des Groundtruth-Tracking als Test des Gesamtsystems in Kapitel 10.4 zeigte, dass die fortlaufende Verfolgung des Roboters über einen kontinuierlich angepassten ROI über eine von *OpenCV* bereitgestellte Fittingfunktion nur eine unzureichende Lösung ist. Die Region des Roboters wird dadurch unzureichend geschätzt und es entstehen fehlerhafte Schätzungen, die nach Möglichkeit vom System korrigiert werden. Dennoch lassen sich mit ein bisschen Wissen über den Aufbau des Versuchs die Trajektorien des Roboters deutlich herausarbeiten, welche durch die Kreuzung der Feldlinien einige Schwankungen besitzen,

und eine Annäherung an die Lokalisierung innerhalb des Root-Mean-Squared-Error aus dem Detektionsfehlerversuch darstellt. Außerdem ist die durchschnittliche Entfernung der berechneten Punkte zum idealen Laufweg geringer als die der Lokalisierung. Dadurch lässt sich das Tracking als annehmbar für die Verfolgung eines Roboters ansehen, insbesondere wenn man Wissen über das eigene Experiment einfließen lässt. Für die vollständige Erfassung eines SPL-Spiels wären weitere Maßnahmen zur Fehlervermeidung wünschenswert.

11.1.4 Die Farbkalibrierung

Die Untersuchung der Farbkalibrierung in Kapitel 10.5 zeigte, dass die initiale Farbkalibrierung sich von der dynamischen Variante kaum unterscheidet, was den validen Anteil der Schätzungen angeht. Jedoch ist das Verhältnis der validen Schätzungen zur erwarteten Anzahl valider Schätzungen über die Bilderanzahl bei der konstanten Farbspezifikation wesentlich besser, da durchgängiger Perzepte für die Schätzungen geliefert wurden, wodurch diese weder abdriften noch gelöscht werden konnten. Dadurch ist eine initiale Farbkalibrierung bei günstigeren Lichtverhältnissen der dynamischen Variante vorzuziehen. Dies weist darauf hin, dass eine automatische Erfassung von günstigen Gelegenheiten zur Rekalibrierung der Farbklassifizierung für das System von Vorteil wäre. Ebenso erwies sich bei Probeläufen der Evaluation die wiederholte Zuweisung der Farbklassen über Referenzfarben bei der dynamischen Variante als nicht praktikabel, weshalb eine Entfernung von den Referenzfarben in zukünftigen Entwicklungen naheliegt.

11.1.5 Die Blickwinkelunabhängigkeit

Das Experiment der Blickwinkelunabhängigkeit in Kapitel 10.6 zeigt, dass trotz unterschiedlicher Aufbauten, die von der Kamerakalibrierung im Sinne der ermittelten Ausrichtung her richtig erfasst wurden, bei dem gleichen Experiment auch ähnliche Ergebnisse herauskommen. Ebenso wurden Grenzfälle der farbbasierten Hinter- und Vordergrunddetektion ersichtlich, sowie Fehler des Trackings durch zu große Roboterregionen, die ganze Liniensegmente umfassen, wie in Abb. 11.1 dargestellt.



Abbildung 11.1 Ein Beispielbild, bei dem ein Liniensegment die Ellipse zur Beschreibung des Roboters verfälscht.

11.1.6 Die Auswirkung von Verdeckungen

Die Evaluation der Auswirkung von Verdeckungen in Kapitel 10.7 zeigte, dass eben solche, wie in der Beschreibung des Detektionsalgorithmus gemutmaßt, deutliche Verfälschungen verursachen, da die Schätzungen immer zu dem Roboter im Vordergrund konvergieren und die Detektion des laufenden Roboters beim Verlassen der Verdeckung langsam erst die richtige Roboterregion annähert. Dadurch besitzt dieses System einen Nachteil, den viele Bildverarbeitungssysteme haben. Jedoch könnte man das Verhalten durch eine ausführlichere Modellierung kompensieren, da bekannt ist, dass die Schätzungen zu der Schätzung hin konvergieren, die näher an der Kamera ist, wobei eine Verdeckung auch frühzeitig vom Robotererkenner angemerkt werden könnte und dabei eine Aktualisierung des hinteren Roboters unterbindet. Analog zu dem Groundtruthtest ließen sich auch hier die tatsächlichen Wege der Roboter leicht aus den Daten extrahieren.

11.1.7 Die Laufzeiten der Anwendung

Die Untersuchung der Laufzeiten in Kapitel 10.8 zeigt, dass im Durchschnitt für eine vollständige Analyse eines Bildes im Sinne von Zugriff, Vorverarbeitung und Auswertung ca 522 Millisekunden benötigt werden. Dabei sind die aufwändigsten Funktionen des Systems, die regelmäßig aufgerufen werden, das Entzerren des Bildes und die Bildsegmentierung. Dabei ist das Erstere eine Operation, die nur einmal auf ein Bild angewandt wird, würde man jedoch nur auf ein Bild zugreifen und es entzerren, würde man in dem System trotzdem nicht einmal sechs Bilder die Sekunde schaffen. Die Segmentierung ist in der Hinsicht sogar noch unangenehmer, da sie die Grundlage für die Vorverarbeitung der Szene ist und ebenso der Analyse im Robotererkenner. Hohe Laufzeiten in dieser Funktion schlagen sich um ein vielfaches im gesamten System nieder. Eine Lösung, um die Laufzeit nach der initialen Kalibrierung deutlich zu verringern, ist eine Berechnung der Farbtabelle, über welche man in der weiteren Analyse segmentieren kann. Dieser Ansatz wurde ausprobiert, in dem man für alle möglichen Pixelwerte eines HSV-Farbraumbildes die Vorhersage des *Gaussian Mixture Model* in eine Tabellenform bringt. Die Laufzeiten dieses Ansatzes sind in Tab. 11.1 analog zu den Werten aus Kapitel 10.8 erfasst und aufgelistet worden.
Die Variante der Farbtabelle benötigt initial 13 Sekunden und ist im durchschnittlichen Fall der Segmentierung um mehr als Faktor 10 schneller. Dies lässt sich auch an Abb. 11.2 ablesen, deren Daten auf denselben Bildern ermittelt wurden wie in Kapitel 10.8.
Eine Lösung zur Erhöhung der allgemeinen Performanz wäre die Einbindung von CUDA in *OpenCV* [OpenCV, 2018], welches durch Parallelisierung und Nutzung der Grafikkarte viele der verwendeten Algorithmen um ein vielfaches beschleunigt. Obwohl Echtzeitfähigkeit, wie sonst im Kontext von B-Human, mit der vollständigen Analyse von 60 Bildern die Sekunde, kein Kriterium für das hier entwickelte System ist, ist eine kurze Dauer für Analysen in der Endanwendung wünschenswert. Ebenso ist es hilfreich zu wissen, wo im ganzen System die meiste Rechenkapazität benötigt wird.

Farbkalibrierung mit Tabelle	Minimum (ms)	Maximum (ms)	Durchschnitt (ms)
Gesamtkalibrierung		13682	
Erstellung der Trainingsmenge		0.18	
Anlernen des GMM		20.94	
Berechnung des Feldausschnitts		25.4	
Berechnung der Hintergrundregionen		5.8	
Segmentierung	0.006	9.53	0.13
Erstellung der Farbtabelle		13604.5	

Tabelle 11.1 Die Laufzeiten der Farbanalyse mit einer Farbtabelle, die aus dem *Gaussian Mixture Model* berechnet wurde.

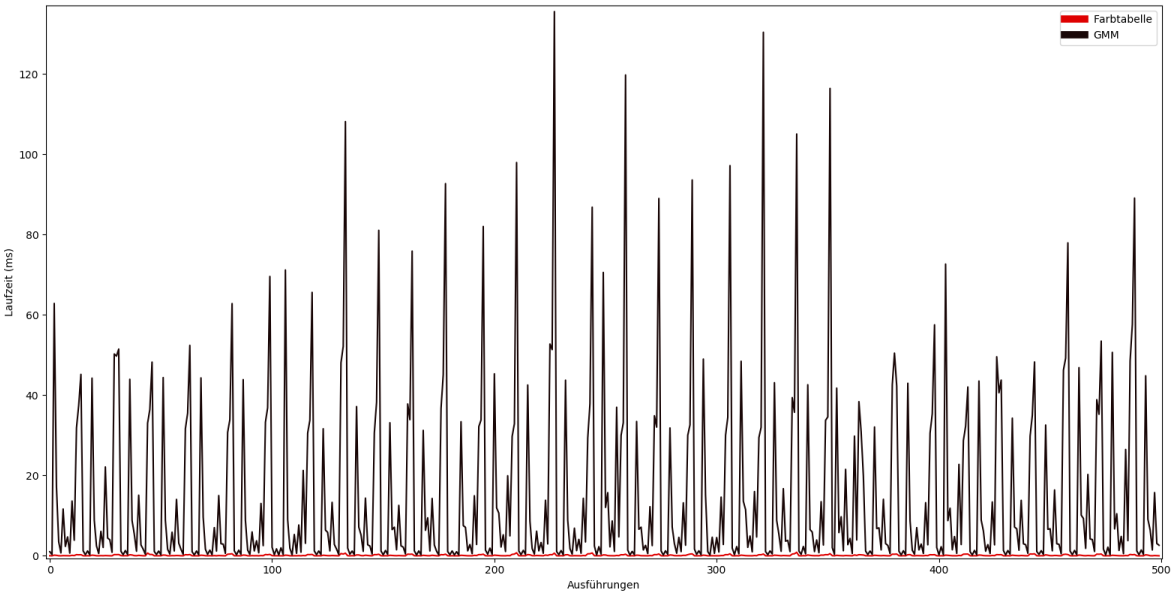


Abbildung 11.2 Laufzeitverhalten der beiden Methoden auf gleichen Daten während der Videoanalyse.

11.1.8 Der Gesamttest

Der Test des Gesamtsystems im Kontext des RoboCups in Kapitel 10.9 zeigt, dass durchaus viele brauchbare Daten während der Analyse gesammelt wurden. Jedoch verfälschen Verdeckungen und False-Positives die Daten und erhöhen stark die Datenmenge. Dadurch wird ersichtlich, dass in die Detektion mehr Arbeit investiert werden muss, um diese Fehler zu verhindern. Ein anderer Ansatz wäre eine bessere Nachverarbeitung der Daten, bei denen zu kurze Trajektorien verworfen werden und offensichtlich zerteilte Trajektorien zusammengeführt werden, was im Rahmen der Evaluation manuell gemacht wurde. Somit ist eine Verwendung im Kontext des RoboCups als einfache Methode, um vollständig automatisiert Spiele auszuwerten, nicht gegeben. Jedoch wurde auch hier gezeigt, dass mit zusätzlichem Wissen

auch sinnvolle Trajektorien herausgearbeitet werden können.

11.2 Ausblick

Eine einfache Erweiterung, die eine umfassendere Schätzung einer Roboterposition bietet, wäre die Entwicklung einer Rotationsdetektion wie von Mühlenbrock und Laue [Im Erscheinen]. Die Grundlagen sind in diesem System schon vorhanden, da sowohl Konturdetektion als auch Farbinformationen benutzt werden. Dadurch müsste eine Erweiterung der Schätzung des Kalman-Filters integriert werden, aber das Resultat wären vollständige Vergleichswerte zur Verifikation, wie es auch in anderen Systemen möglich ist. Dies geschieht auf reiner Detektionsbasis auch in dem Visionsystem der Small Size League von Zickler u. a. [2010].
Außer Ansätzen, die eine direkte Einbindung in das System erfordern, könnte man die Analyse der generierten Daten erweitern. Die in dieser Arbeit vorgestellten anschließenden Auswertungen wurden mit verschiedenen Python-Skripten vorgenommen, die hier primär zur Veranschaulichung der Daten dienten oder gar eine statistische Auswertung ermöglichten. In Zukunft könnte man durch diese anschließende Verarbeitung auch Ausreißer in den Trajektorien oder gar Inkonsistenzen herausfiltern. Dadurch würde keine fehleranfällige Selektion der Daten von einem Menschen durchgeführt werden müssen. Diese Nachbearbeitung kann auch dadurch unterstützt werden, dass man den generierten Daten einen Zeitstempel aus dem Video zuweist.
Wie in Kapitel 8.4 bereits diskutiert, hat man in der zukünftigen Weiterentwicklung des Robotererkenners viele Möglichkeiten. Eine Variante wäre es, die Anwendbarkeit der Verdeckungsdetektion von Otake, Murakami und Naruse [2008] zu überprüfen. Die dortige Entwicklung wurde auf die RoboCup Small Size League zugeschnitten, aber besitzt einige Gemeinsamkeiten mit der hier vorgestellten Arbeit. Die Roboter werden von einer Kamera detektiert, die möglichst das ganze Feld erfassen soll, die Perspektive ermöglicht dabei Verdeckungen, die maßgeblich die Detektion erschweren, und die initiale Klassifizierung wird über Farben ermöglicht. Verdeckungskandidaten werden dann über *Higher Order Local Autocorrelations* (HLAC) Merkmale und *Support Vector Machines* (SVM) als teilweise verdeckt oder als Störung klassifiziert. Die Autoren haben angemerkt, dass das Verfahren nach Konstruktion robust gegenüber Beleuchtungsveränderungen sein soll, es ist jedoch noch offen, ob dieser Ansatz in einer anderen Perspektive und einem anderen primären Detektionsziel ebenfalls funktioniert. Eine andere Möglichkeit, Verdeckungen zu behandeln, ist es, in der Modellierung und über die Regionen der Roboter zukünftige Verdeckungen vorherzusagen. Mit dem Wissen aus der Evaluation, dass bei einer Verdeckung immer nur der vordere Roboter detektiert wird, kann man in der Modellierung dafür sorgen, dass die Messung nur für den Roboter verwendet wird, der näher an der Kamera ist, und dabei die zweite Messung unterbinden, da sie langsam zum falschen Roboter konvergiert.
Eine andere mögliche Erweiterung besteht darin, die Untersuchung der Highlight-Erkennung, wie von Ren und Jose [2009], für die SPL durchzuführen. Dabei müssten mehr Informatio-

nen aus dem Videomaterial gezogen werden und für die *Attention*-Berechnung ausgewertet werden. Dabei wäre die Untersuchung in diesem System gerade interessant, da im visuellen Bereich kein Einfluss durch den Zuschauerraum entsteht und im Audiobereich in der Regel kein Einfluss durch Kommentatoren oder Zuschauer genommen wird.
Eine zusätzliche Erweiterung im Bereich der Erkennung wäre eine Lagedetektion, die angibt, ob der gefundene Roboter steht oder gerade auf dem Feld liegt. Durch diese Information könnte mehr Wissen in das Tracking und in die Modellierung einfließen, wodurch präzisere Vorhersagen und damit bessere Daten erzielt werden können. Ein Nachteil, der daraus erwächst, ist der wahrscheinliche Verlust der Blickwinkelunabhängigkeit und der damit einhergehenden Positionsunabhängigkeit der Kamera. Allgemein wäre auch eine höhere Detektionsrate bei der Robotererkennung wünschenswert, wobei dafür auch maschinelles Lernen verwendet werden kann. Dabei kann dieses System zum Generieren der Trainingsdaten benutzt werden kann.
Im Bereich der Kamerakalibrierung wäre eine Automatisierung für die Bestimmung der extrinsischen Kameraparameter der nächste sinnvolle Schritt, da dadurch bei bekannten intrinsischen Parametern keine manuellen Kalibrierungsschritte mehr durchgeführt werden müssen, welche zu einer gewissen Fehleranfälligkeit neigen. Hinzu kommt, dass dadurch die Benutzung des Systems für fachfremde Anwender vereinfacht werden würde. Eine Variante wäre ein Modelfitting wie von Egorova u. a. [2005], bei dem iterativ die richtigen Parameter gefunden werden, um die Feldlinien korrekt in das Bild zu projizieren.
Die ideale Erweiterung wäre ein allgemein anwendbares Werkzeug im RoboCup, welches ähnlich zum Small Size League Vision System von Zickler u. a. [2010] mehrere Kameras unterstützt, aber eine komplett automatisierte Kalibrierung wie bei Egorova u. a. [2005] enthält und dabei eine vollständige Detektion (Position und Orientierung) ohne Marker ermöglicht.

11.3 Fazit

Das im Rahmen dieser Arbeit vorgestellte und evaluierte System zeigt, dass zum externen Erfassen eines gelaufenen Weges eines *NAO*s kein kostspieliges Trackingsystem verwendet werden muss. Einfache Videodateien können somit offline analysiert werden und die Daten anschließend ausgelesen werden. Die Präzision der Daten lässt sich mit der von anderen Systemen vergleichen mit einer für Versuchszwecke akzeptablen Reichweite. Trotz diverser Phänomene beim Erfassen der Roboter lassen sich viele Experimente gut über die generierten Daten nachvollziehen. Jedoch erlaubt der hier verwendete Detektionsansatz keine absolut präzise Analyse der Spiele der SPL im RoboCup, da durch Verdeckungsprobleme und False-Positives die Schätzungen stark verfälscht werden. Dennoch kann es als Referenzsystem im kleineren Maßstab benutzt und durch weitere Entwicklungen, wie im Ausblick beschrieben, verbessert werden.

11.4 Danksagung

Diese Arbeit ist im Umfeld des Teams B-Human entstanden und wurde von der Arbeitsgruppe Multisensorische Interaktive Systeme der Universität Bremen mit der Bereitstellung der *GoPro* sowie durch viele Ratschläge unterstützt. Danke auch an René Wagner, der mir die korrekte Kalibrierung der *GoPro* außerhalb von *OpenCV* ermöglicht hat. Ohne euch wäre die Arbeit nicht in dieser Form entstanden.

Literatur

Burger, Wilhelm und Mark James Burge (2006). *Digitale Bildverarbeitung - Eine Einführung mit Java und ImageJ*. Springer. ISBN: 3-540-21465-8.

Choi, Euisun und Chulhee Lee (2003). "Feature extraction based on the Bhattacharyya distance". In: *Pattern Recognition* 36.8, S. 1703–1709.

Egorova, Anna u. a. (2005). "Plug and play: Fast automatic geometry and color calibration for cameras tracking robots". In: *RoboCup 2004: Robot Soccer World Cup VIII*. Hrsg. von Daniele Nardi u. a. Bd. 3276. Lecture Notes in Artificial Intelligence. Springer, S. 394–401.

Eigen (2017a). *Linear algebra and decompositions*. `http://eigen.tuxfamily.org/dox/group__TutorialLinearAlgebra.html` Abgerufen am: 10.11.2017.

— (2017b). *Main Page*. `http://eigen.tuxfamily.org/index.php?title=Main_Page` Abgerufen am: 10.11.2017.

Frese, Udo (2017). *Echtzeitbildverarbeitung - ebvfrese1309.pdf*. Vorlesungsfolien an der Universität Bremen - `https://svn-agbkb.informatik.uni-bremen.de/ufrese/teaching/ebv/slides/ebvpublic/slides/` Abgerufen am: 8.3.2018.

GoPro (2017). *GoPro Hero 3+ Black Edition - Benutzerhandbuch*. `https://gopro.com/content/dam/help/hero3plus-black-edition/manuals/HERO3_Plus_Black_UM_GER_REVD.pdf` Abgerufen am: 10.11.2017.

Gunnarson, Ketill, Fabian Wiesel und Raúl Rojas (2006). "The Color and the Shape: Automatic On-Line Color Calibration for Autonomous Robots". In: *RoboCup 2005: Robot Soccer World Cup IX*. Hrsg. von Ansgar Bredenfeld u. a. Bd. 4020. Lecture Notes in Artificial Intelligence. Springer, S. 347–358.

Hartley, Richard und Andrew Zisserman (2011). *Multiple View Geometry in computer vision*. 2. Aufl. 9th Printing. Cambridge University Press. ISBN: 978-0-521-54051-3.

Jochmann, Gregor u. a. (2012). "Efficient Multi-hypotheses Unscented Kalman Filtering for Robust Localization". In: *RoboCup 2011: Robot Soccer World Cup XV*. Hrsg. von Thomas Röfer u. a. Bd. 7416. Lecture Notes in Artificial Intelligence. Springer, S. 222–233.

Kaufmann, Ulrich u. a. (2005). "Visual Robot Detection in RoboCup Using Neural Networks". In: *RoboCup 2004: Robot Soccer World Cup VIII*. Hrsg. von Daniele Nardi u. a. Bd. 3276. Lecture Notes in Artificial Intelligence. Springer, S. 262–273.

Khandelwal, Piyush und Peter Stone (2012). “A Low Cost Ground Truth Detection System for RoboCup Using the Kinect”. In: *RoboCup 2011: Robot Soccer World Cup XV*. Hrsg. von Thomas Röfer u. a. Bd. 7416. Lecture Notes in Artificial Intelligence. Springer, S. 515–526.

Kwok, Cody und Dieter Fox (2005). “Map-Based Multiple Model Tracking of a Moving Object”. In: *RoboCup 2004: Robot Soccer World Cup VIII*. Hrsg. von Daniele Nardi u. a. Bd. 3276. Lecture Notes in Artificial Intelligence. Springer, S. 18–33.

Misu, Toshie u. a. (2009). “Probabilistic integration of tracking and recognition of soccer players”. In: *Advances in Multimedia Modeling*. Hrsg. von Benoit Huet u. a. Bd. 5371. Lecture Notes in Computer Science. Springer, S. 39–50.

Mühlenbrock, Andre und Tim Laue (Im Erscheinen). “Vision-based Orientation Detection of Humanoid Soccer Robots”. In: *RoboCup 2017: Robot World Cup XXI*. Lecture Notes in Artificial Intelligence. `https://www.b-human.de/downloads/publications/2017/OrientationDetection.pdf` Abgerufen am: 2.1.2018. Springer, Unbekannt.

Nisticò, Walter u. a. (2007). “Cooperative visual tracking in a team of autonomous mobile robots”. In: *RoboCup 2006: Robot Soccer World Cup X*. Hrsg. von Gerhard Lakemeyer u. a. Bd. 4434. Lecture Notes in Artificial Intelligence. Springer, S. 146–157.

OpenCV (2016). *Expectation Maximization*. `https://docs.opencv.org/2.4.13.2/modules/ml/doc/expectation_maximization.html` Abgerufen am: 4.12.2017.

— (2017a). *Camera Calibration and 3D Reconstruction*. `https://docs.opencv.org/3.3.0/d9/d0c/group__calib3d.html` Abgerufen am: 10.11.2017.

— (2017b). *Introduction*. `https://docs.opencv.org/3.3.1/d1/dfb/intro.html` Abgerufen am: 10.11.2017.

— (2018). *CUDA*. `https://opencv.org/platforms/cuda.html` Abgerufen am: 1.3.2018.

Otake, Kazutoki, Kazuhito Murakami und Tadashi Naruse (2008). “Precise Extraction of Partially Occluded Objects by Using HLAC Features and SVM”. In: *RoboCup 2007: Robot Soccer World Cup XI*. Hrsg. von Ubbo Visser u. a. Bd. 5001. Lecture Notes in Artificial Intelligence. Springer, S. 17–28.

Paar, Christof und Jan Pelzl (2010). *Understanding Cryptography - A Textbook for Students and Practitioners*. Springer. ISBN: 978-3-642-44649-8.

Qt (2017). *What is Qt*. `https://www.qt.io/what-is-qt/` Abgerufen am: 10.11.2017.

Ren, Reede und Joemon M Jose (2009). “General highlight detection in sport videos”. In: *Advances in Multimedia Modeling*. Hrsg. von Benoit Huet u. a. Bd. 5371. Lecture Notes in Computer Science. Springer, S. 27–38.

Reynolds, Douglas (2015). “Gaussian mixture models”. In: *Encyclopedia of biometrics*, S. 827–832.

RoboCup Technical Committee (2017). *RoboCup Standard Platform League (NAO) Rule Book*. `http://spl.robocup.org/wp-content/uploads/downloads/Rules2017.pdf` Abgerufen am: 16.11.2017.

Rodrigues, Joao u. a. (2014). “A computer vision based web application for tracking soccer players”. In: *Universal Access in Human-Computer Interaction*. Hrsg. von Constantine

Stephanidis und Margherita Antona. Bd. 8513. Lecture Notes in Computer Science. Springer, S. 450–462.

Röfer, Thomas u. a. (2017). *B-Human Team Report and Code Release 2017*. Only available online: https://github.com/bhuman/BHumanCodeRelease/tree/coderelease2017.

Rojas, Raul und Alexander Gloye Förster (2006). "Holonomic control of a robot with an omnidirectional drive". In: *KI-Künstliche Intelligenz* 20.2, S. 12–17.

Rosten, Edward und Tom Drummond (2006). "Machine learning for high-speed corner detection". In: *Computer Vision–ECCV 2006*, S. 430–443.

Ruiz-del-Solar, Javier, Patricio Loncomilla und Paul Vallejos (2007). "An Automated Refereeing and Analysis Tool for the Four-Legged League". In: *RoboCup 2006: Robot Soccer World Cup X*. Hrsg. von Gerhard Lakemeyer u. a. Bd. 4434. Lecture Notes in Artificial Intelligence. Springer, S. 206–218.

Sheh, Raymond und Geoff West (2005). "Visual Tracking and Localization of a Small Domestic Robot". In: *RoboCup 2004: Robot Soccer World Cup VIII*. Hrsg. von Daniele Nardi u. a. Bd. 3276. Lecture Notes in Artificial Intelligence. Springer, S. 410–417.

SoftBank Robotics. *Find out more about NAO*. https://www.ald.softbankrobotics.com/en/robots/nao/find-out-more-about-nao Abgerufen am: 14.11.2017.

— *NAO - Developer Guide - Construction*. http://doc.aldebaran.com/2-4/family/robots/dimensions_robot.html Abgerufen am: 14.11.2017.

— *NAO - Developer Guide - Motherboard*. http://doc.aldebaran.com/2-4/family/robots/motherboard_robot.html Abgerufen am: 14.11.2017.

— *NAO - Developer Guide - Video camera*. http://doc.aldebaran.com/2-4/family/robots/video_robot.html Abgerufen am: 14.11.2017.

Suppa, Michael (2017). *Semantic 3D Perception - 3DPerception_l5_WS17-18.pdf*. Vorlesungsfolien an der Universität Bremen.

Szeliski, Richard (2010). *Computer Vision: Algorithms and Applications*. Draft-Version von http://szeliski.org/Book, Version 'September 3. 2010' abgerufen am 20.12.17. Springer.

Team B-Human (2015). *B-Human, Willkommens-Seite*. https://www.b-human.de/index-de.html Abgerufen am: 10.11.2017.

Teschl, Gerald und Susanne Teschl (2007). *Mathematik für Informatiker - Band 2 - Analysis und Statistik*. 2. Aufl. Springer. ISBN: 978-3-540-72451-3.

Tetragramm auf GitHub (2017). *PositionCalculatorImpl*. https://github.com/Tetragramm/opencv_contrib/blob/master/modules/mapping3d/src/positionCalc.cpp#L130 Abgerufen am: 11.12.2017.

The RoboCup Federation (2016a). *RoboCup History*. http://www.robocup.org/a_brief_history_of_robocup Abgerufen am: 13.11.2017.

— (2016b). *RoboCup Objective*. http://www.robocup.org/objective Abgerufen am: 13.11.2017.

— (2016c). *Soccer Standard Platform League*. http://www.robocup.org/leagues/5 Abgerufen am: 14.11.2017.

Thrun, Sebastian, Wolfram Burgard und Dieter Fox (2006). *Probabilistic Robotics*. The MIT Press. ISBN: 978-0-262-20162-9.

Weigel, David (2015). "Videobasierrte Lokalisierung von Hallenfußballspielern". Magisterarb. Deutschland: Universität Bremen.

Wilking, Dirk und Thomas Röfer (2005). "Realtime Object Recognition Using Decision Tree Learning". In: *RoboCup 2004: Robot Soccer World Cup VIII*. Hrsg. von Daniele Nardi u. a. Bd. 3276. Lecture Notes in Artificial Intelligence. Springer, S. 556–563.

Zeppenfeld, Klaus (2004). *Lehrbuch der Grafikprogrammierung - Grundlagen, Programmierung, Anwendung*. 1. Aufl. Spektrum Akademischer Verlag. ISBN: 3-8274-1028-2.

Zickler, Stefan u. a. (2010). "SSL-Vision: The Shared Vision System for the RoboCup Small Size League". In: *RoboCup 2009: Robot Soccer World Cup XIII*. Hrsg. von Jacky Baltes u. a. Bd. 5949. Lecture Notes in Artificial Intelligence. Springer, S. 425–436.