

**Universität Leipzig**

Institut für Informatik

Data Science Master

Masterarbeit

zur Erlangung des akademischen Grades

**Master of Science (M.Sc.)**

**Entwicklung eines kamerabasierten  
Tracking-Systems für die Roboterplattform  
Dezibot mithilfe von OLED-Bildschirmen**

Eingereicht von: Miriam Louise Carnot

Matrikelnummer: 3736052

Leipzig 3. Januar 2023

Betreuer: Prof. Dr. Martin Middendorf  
M.Sc. Tobias Jagla



# **Selbstständigkeitserklärung**

Ich versichere, dass die Masterarbeit mit dem Titel „Entwicklung eines kamerabasierten Tracking-Systems für die Roboterplatform Dezibot mithilfe von OLED-Bildschirmen“ nicht anderweitig als Prüfungsleistung verwendet wurde und diese Masterarbeit noch nicht veröffentlicht worden ist. Die hier vorgelegte Masterarbeit habe ich selbstständig und ohne fremde Hilfe abgefasst. Ich habe keine anderen Quellen und Hilfsmittel als die angegebenen benutzt. Diesen Werken wörtlich oder sinngemäß entnommene Stellen habe ich als solche gekennzeichnet.

Leipzig, 3. Januar 2023

Unterschrift



# Kurzfassung

Zur gründlichen Erforschung von schwarmintelligentem Verhalten von Robotern ist eine Verfolgung der Roboter während der Durchführung dies bezogener Experimente von Nöten. Für die Versuche wurden kleine Roboter entwickelt, sogenannte Dezibots. Sie wurden vor allem aufgrund ihres geringen Preises bevorzugt. Die Dezibots befinden sich bei den Experimenten in einer speziell angefertigten Arena, die zwei mal ein Meter groß ist. Um die Versuche detailliert zu dokumentieren und zu verfolgen, sollen für jeden Zeitschritt die Positionen, Orientierungen und Roboternummern aller Roboter festgestellt werden, die sich aktuell in der Arena befinden. Jeder Roboter hat eine eigene Nummer, die sich nicht ändert. Der Beitrag dieser Arbeit umfasst zum Einen das Auffinden der Arena-Ecken, der Homographie des aufgenommenen Bildes und dem Ausfindigmachen der Roboter. Des Weiteren werden Barcodes zur eindeutigen Bestimmung der Roboter entwickelt. Diese Barcodes werden auf den OLED-Bildschirmen der Roboter angezeigt. Der Vorteil dieser Bildschirme ist, dass sie sowohl bei Tageslicht als auch im Dunkeln eingesetzt werden können. Bei jedem gefundenen Roboter werden seine Orientierung und seine Roboternummer anhand des auf dem Display gezeigten Barcodes bestimmt. Schlussendlich soll die Laufzeit bestmöglich verbessert werden. Von Interesse sind vor allem die Analyse von Live-Präsentationen und Videos. Aber auch die Behandlung von Bilderreihen und einzelnen Bildern ist durch das Tracking-System möglich. Die Aufnahme erfolgt mit einer 4k-Kamera, die circa 1,5 Meter über der Arena platziert wurde.



# Inhaltsverzeichnis

<b>1 Einleitung</b>	<b>1</b>
<b>2 Theoretische Grundlagen</b>	<b>5</b>
2.1 Methoden der Bildverarbeitung . . . . .	5
2.1.1 Grundlagen der Kamerakalibrierung . . . . .	6
2.1.2 Grundlagen der Homographie . . . . .	7
2.1.3 Objekterkennung und -verfolgung . . . . .	7
2.2 Barcodes . . . . .	8
2.3 Laufzeitanalyse durch Code Profiling . . . . .	9
2.4 Grundlagen der Parallelverarbeitung . . . . .	9
<b>3 Methodenentwicklung</b>	<b>11</b>
3.1 Getroffene Annahmen . . . . .	11
3.2 Kamerakalibrierung . . . . .	13
3.3 Exakte Darstellung der Arena . . . . .	13
3.3.1 Lokalisierung der Arena-Ecken . . . . .	14
3.3.2 Homographie . . . . .	17
3.4 Lokalisierung der Roboter . . . . .	18
3.4.1 Bedingungen zur Roboteridentifizierung . . . . .	19
3.4.2 Handhabung von Überschneidungen . . . . .	21
3.4.3 Festlegen der Roboterbildausschnitte . . . . .	23
3.5 Orientierung der Roboter . . . . .	23
3.5.1 Approximierung des gelben Balkens . . . . .	24
3.5.2 Berechnung des Orientierungswinkels . . . . .	25

## *Inhaltsverzeichnis*

3.6 Design der Barcodes . . . . .	27
3.6.1 Barcode-Design 1 . . . . .	28
3.6.2 Barcode-Design 2 . . . . .	29
3.7 Dekodierung der Barcodes . . . . .	30
3.7.1 Erstellung von Abtastlinien . . . . .	31
3.7.2 Auslesen der Abtastlinien . . . . .	32
3.7.3 Dekodierung jeder Abtastlinie . . . . .	33
3.7.4 Abstimmung aller Abtastlinien . . . . .	35
3.8 Überarbeitung der Ergebnisse . . . . .	37
3.9 Anzeigen und Protokollieren der Ergebnisse . . . . .	38
3.10 Laufzeitanalyse mithilfe von Profilern . . . . .	39
3.11 Umsetzung der Parallelverarbeitung . . . . .	41
<b>4 Resultate</b>	<b>43</b>
4.1 Präzision der Positionsbestimmung . . . . .	43
4.1.1 Erkennungsfehler bei der Positionsbestimmung . . . . .	43
4.1.2 Beständigkeit der Positionsbestimmung . . . . .	45
4.2 Präzision der Berechnung der Orientierung . . . . .	45
4.2.1 Erkennungsfehler bei der Berechnung der Orientierung . . . . .	45
4.2.2 Beständigkeit bei der Berechnung der Orientierung . . . . .	47
4.3 Präzision und Zuversichten der Barcode-Dekodierung . . . . .	47
4.3.1 Präzision der Barcode-Dekodierung . . . . .	48
4.3.2 Zuversichten der erkannten Roboternummern . . . . .	50
4.4 Performanz des Tracking-Systems . . . . .	54
<b>5 Diskussion</b>	<b>57</b>
<b>6 Fazit und Ausblick</b>	<b>59</b>
<b>Abbildungsverzeichnis</b>	<b>61</b>
<b>Tabellenverzeichnis</b>	<b>63</b>
<b>Literatur</b>	<b>65</b>

# 1 Einleitung

Schwarmintelligenz bezeichnet ein Forschungsgebiet, das sich mit Algorithmen beschäftigt, die das kollektive Verhalten von Insektenkolonien nachstellen sollen [1, 5, 21, 35]. Seit seinem Beginn in den späten 1980er-Jahren wurde eine Vielzahl von ebendiesen Algorithmen entwickelt [6].

Von Interesse ist neben der theoretischen Entwicklung und Simulation der Vorgänge auch die praktische Umsetzung mithilfe von Schwarmrobotern [4, 11, 29]. Der rapide Ausbau der Robotik in den letzten Jahren ermöglicht eine günstigere Herstellung von Robotern und damit eine simplere Anwendung von diesen in wissenschaftlichen Experimenten sowie in industriellem Umfeld [19, 31, 37, 41, 42].

Als Schwarmrobotik bezeichnet man ein Forschungsgebiet, bei dem mehrere Roboter miteinander vernetzt einen Schwarm nachbilden. Sie kann überall da zum Einsatz kommen, wo es von Vorteil ist, dass sich die Roboter lokal verständigen ohne eine Zentrale, die Befehle gibt. Zu lokaler Verständigung gehören beispielsweise Pheromone, die Ameisen in der Natur ablegen und die anderen dabei helfen, den richtigen Weg zu finden [24, 36]. Eine übergeordnete Zentrale bietet in jedem Fall eine große Angriffsfläche und es ist notwendig, dass jeder Roboter Verbindung zur Zentrale aufnehmen kann, was bei weiten Entfernung nicht selbstverständlich ist. Ein solches Szenario ist sowohl beim Überwachen als auch beim Erkunden neuer Gebiete denkbar.

In der Forschung sollen das Verhalten und die Kommunikation der Roboter getestet werden. Das ScaDS.AI in Leipzig hat zu diesem Zwecke eine Arena bauen lassen, auf der mit sogenannten Dezibots experimentiert werden soll. Dezibots sind kleine, günstige Roboter mit grundlegenden Sensoren und Fortbewegungsfähigkeiten [12, 13]. Die Besonderheit der Dezibots ist ihre Fähigkeit, UV-Licht auszusenden, welches die Photoswitch-Schicht der speziell entwickelten Arena-Oberfläche anregt und

## KAPITEL 1. EINLEITUNG

seine Farbe ändern lässt. Wie auch Pheromone verblassen diese über die Zeit. Über Sensoren können die Roboter die Farbe auf der Arena erkennen und somit untereinander lokal kommunizieren. Eine weitere Arena der Universität Leipzig nutzt eine Phosphor-Schicht, die durch das UV-Licht zum Leuchten gebracht wird. Auch hier klingt das Licht über die Zeit ab. Diese Arena hat die Besonderheit, dass sie nur im Dunkeln für Versuche genutzt werden kann.

Um zukünftige Experimente besser zu analysieren, wird in dieser Arbeit ein Tracking-System für Dezibots entwickelt, welches die Positionen, die Orientierungen und die Roboternummern aller auf der Arena befindlichen Dezibots ausfindig machen soll. Die Positionen der Roboter sollen so angegeben werden, dass der Koordinatenursprung genau in der Mitte der Arena liegt. Beide Achsen sind einem Standard-Koordinatensystem entsprechend gerichtet. Die Orientierungen werden als Winkel dargestellt, die gegen den Uhrzeigersinn zunehmen, entsprechend dem mathematisch positivem Sinn. Jeder Roboter hat eine eigene Nummer, die sich nicht ändert.

Die Verfolgung geschieht primär über einen OLED-Bildschirm (Organische Leuchtdiode, englisch: organic light-emitting diode) [38]. Dieser ist an der Oberseite eines jeden Roboters befestigt. Für die Bildschirme wurden zwei Barcode-Designs entworfen, die die Binärschreibweise der Roboternummer in zwei verschiedenen Ausführungen darstellen. Die Barcodes codieren sieben Bits. Mit dieser Anzahl können 128 Zahlen repräsentiert werden. Aktuell gibt es circa 30 Dezibots am ScaDS.AI und an der Universität Leipzig, weitere 40 wurden bestellt. Mit lediglich sechs Bits könnten nicht alle Roboter eine eindeutige Roboternummer erhalten. Es ist unwahrscheinlich, dass in Zukunft mehr als 128 Dezibots gleichzeitig auf der zwei Mal ein Meter großen Arena fahren werden. Aus diesem Grund sind sieben Bits ausreichend.

Die zu beantwortenden Forschungsfragen umfassen Folgende:

1. Ist es möglich, die Positionen, Orientierungen und Roboternummern der Roboter allein über ihre OLED-Bildschirme zu verfolgen?
2. Mit welcher Präzision werden diese drei Kriterien erfüllt?
3. Können Werkzeuge wie Parallelverarbeitung oder Profiler die Laufzeit signifikant verbessern?

Die Arbeit ist wie folgt gegliedert: Zuerst werden in Kapitel 2 theoretische Grundlagen geklärt, danach wird die Methodenentwicklung in Kapitel 3 behandelt. Anschließend werden die Resultate in Kapitel 4 offengelegt, und schließlich folgen Diskussion in Kapitel 5 und ein Fazit durch Kapitel 6.



## 2 Theoretische Grundlagen

In diesem Kapitel wird dem Leser theoretisches Hintergrundwissen zum besseren Verständnis der Folgenden näher gebracht. Dabei geht es vor allem darum, welche Methoden der Bildverarbeitung genutzt werden können, um Bilder vor zu verarbeiten und Objekte zu verfolgen. Des Weiteren werden Barcodes vorgestellt, welche hilfreich sind, um Gegenstände eindeutig zu identifizieren. Danach wird die Laufzeitanalyse durch Code Profiling elementar erläutert. Abschließend werden Grundlagen der Parallelverarbeitung beschrieben. Diese ermöglicht es, Teile des Codes auf verschiedenen Prozessoren (englisch: Central Processing Unit, CPU) berechnen zu lassen, um die Laufzeit zu reduzieren.

### 2.1 Methoden der Bildverarbeitung

Für den in der Arbeit beschriebenen Anwendungsfall sind Funktionen zum Öffnen und Anzeigen von Bildmaterial, Kalibrierungsoptionen und das Auffinden von Konturen in einem Bild von Interesse. Die Bibliothek OpenCV stellt verschiedene Methoden der Bildverarbeitung frei zur Verfügung und wurde deshalb als Hilfsmittel genutzt.<sup>1</sup> Culjak et al. geben in ihrem Beitrag eine allgemeine Übersicht zu dieser Bibliothek [9].

---

<sup>1</sup><https://opencv.org/about/>

## KAPITEL 2. THEORETISCHE GRUNDLAGEN

### 2.1.1 Grundlagen der Kamerakalibrierung

Die Kalibrierung der Kamera ist ein essenzieller Schritt, um die verursachte Verzerrung herauszurechnen [27]. Nur so können die Standorte der Roboter später akkurat bestimmt werden. Für die Kalibrierung ist eine Erfassung der Kameraeigenschaften von Nöten. Kalibrierungsmatrizen, auch Checkerboards genannt, können ausgedruckt und dafür verwendet werden [14, 26]. Abbildung 2.1 zeigt eine 10x7 große Kalibrierungsmatrix.

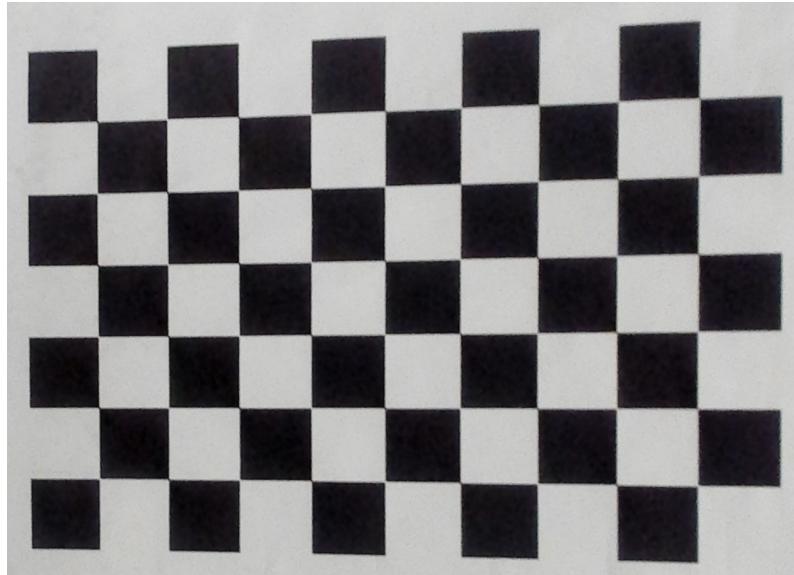


ABBILDUNG 2.1: Beispiel einer Kalibrierungsmatrix zur Kamerakalibrierung.

Eine solche Kalibrierungsmatrix wird platziert und aus verschiedenen Perspektiven mit der zu kalibrierenden Kamera aufgenommen. Alternativ kann die Kamera auch fest stationiert und die Kalibrierungsmatrix bewegt werden. Diese Aufnahmen ermöglichen die Berechnung der Kameraeigenschaften. Die Eigenschaften umfassen die Kameramatrix (ermittelt aus Brennweite und optischem Zentrum), Verzerrungskoeffizienten und Vektoren der Rotation und Translation. Anhand dieser lässt sich die Verzerrung für jegliche Bilder, die mit dieser Kamera aufgenommen werden, herausrechnen.

## 2.1. METHODEN DER BILDVERARBEITUNG

### 2.1.2 Grundlagen der Homographie

Um die Koordinaten in einen bestimmten Ausschnitt zu projizieren, kann man eine Homographie (englisch: homography) nutzen. Homographie bezeichnet auf dem Gebiet der Bildverarbeitung die Zuordnung von Punkten. Sie verbindet Punkte der Bilder aus verschiedenen Ansichten, welche am gleichen Ort liegen [10, 40]. Sie wird daher auch perspektivische Transformation (englisch: Perspective Transformation) genannt, da sie die Transformation einer Ebene in eine andere vornimmt.

Die Koordinaten auf der Arena sollen mit denen im Programm übereinstimmen, daher müssen die Arenaecken den Bildecken zugeordnet werden. Alle anderen Bildpunkte werden entsprechend mit verschoben.

### 2.1.3 Objekterkennung und -verfolgung

Eine Vielzahl an „Tracking“ Frameworks und Merkmalsdetektoren (englisch: feature detectors) zum Auffinden und Verfolgen von Objekten wurde bereits entwickelt [7, 20]. Beliebte Algorithmen zur Objekterkennung basieren auf tiefen neuronalen Netzen [16, 22, 34] oder faltenden neuronalen Netzen (englisch: Convolutional Neural Networks, CNN) [15, 30, 32].

Die neuronalen Netze identifizieren Merkmale der Objekte und verfolgen diese. Elementare Objektmerkmale können auch manuell definiert werden.

Diese Merkmale können sein:

- der Objektumfang (Konturenlänge),
- die Objektgröße,
- die Objektfarben,
- das Begrenzungsrechteck um das Objekt (engl.: bounding box),
- uvm.

Die Objekte können in jedem Einzelbild anhand der Merkmale identifiziert werden und somit über die Zeit sehr genau verfolgt werden.

## KAPITEL 2. THEORETISCHE GRUNDLAGEN

### 2.2 Barcodes

Zur Identifizierung von Objekten werden diese oft mit so genannten Barcodes versehen. Besonders häufig trifft man diese beim Einkaufen oder beim Versenden von Paketen an. Viele dieser Codes können selbst mit herkömmlichen Handykameras und entsprechender Software ausgelesen werden [39]. Aktuell sind vor Allem zweidimensionale Barcodes wie zum Beispiel QR-Codes weit verbreitet [33].

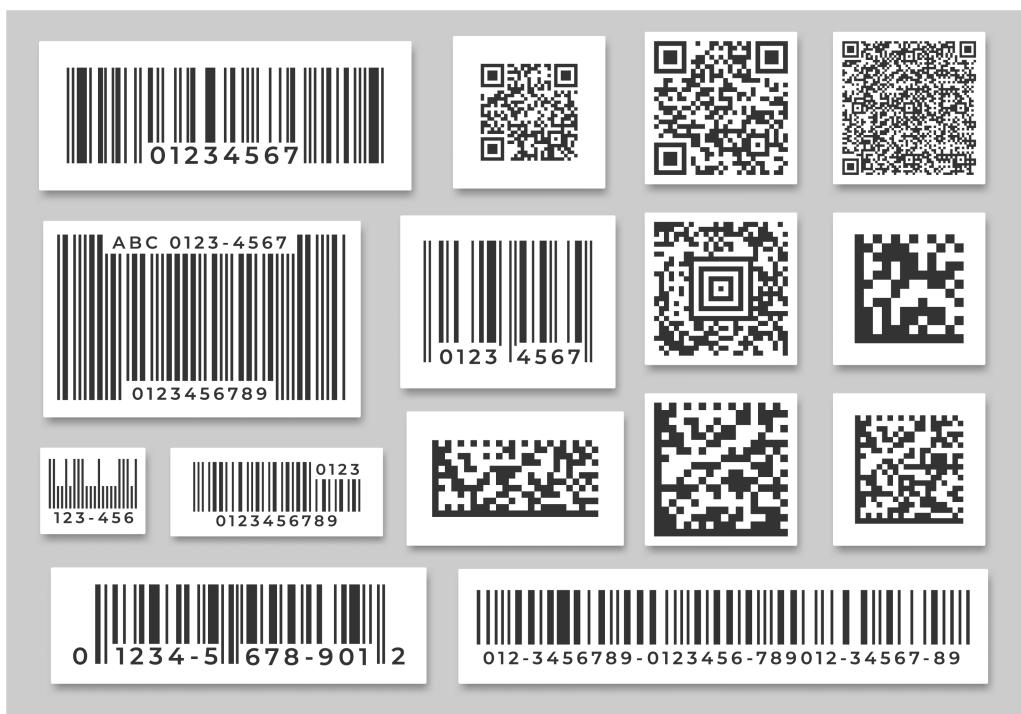


ABBILDUNG 2.2: Verschiedene Barcode-Standards.<sup>2</sup>

Das Ziel eines Barcodes besteht darin, Daten in einem Muster so zu kodieren, dass man diese mithilfe eines Scanners oder einer Kamera auslesen kann.

Aufgrund der Heterogenität der Anwendungsgebiete wurden verschiedene Barcodes und Standards entwickelt, um die Daten von Interesse zu repräsentieren. Diese Daten können in Umfang und Datentypen variieren. Abbildung 2.2 zeigt herkömmliche Barcode-Standards.

<sup>2</sup>Internetquelle Kogoi, Robert: „How to identify Barcodes visually?“, veröffentlicht am 21.05.2022, URL: [www.asp.com.au/how-to-identify-barcode-types-visually](http://www.asp.com.au/how-to-identify-barcode-types-visually), Abruf am 08.12.2022.

## 2.3 Laufzeitanalyse durch Code Profiling

Die Laufzeit eines Programms kann mithilfe von Code Profilern analysiert und entsprechend verbessert werden [23]. Ein Profiler durchläuft das gesamte Programm und stellt fest, welche Funktionen in dem Programm wie oft aufgerufen wurden und wie viel Zeit die Berechnung der Funktion dauerte.

Die fertige Analyse kann mit der Python Bibliothek „Dot Viewer“ visuell aufbereitet werden. Darin wird auch angezeigt, wo die einzelnen Funktionen aufgerufen wurden. Um die Laufzeit zu verbessern, können Funktionsaufrufe bei häufig aufgerufenen Funktionen reduziert werden. Es ist außerdem sinnvoll, den Rechenaufwand von aufwendigen Funktionen bestmöglich zu reduzieren, zum Beispiel indem man eine andere Funktion stattdessen nutzt. Oft ist es möglich, einmal berechnete Ergebnisse wiederholt zu verwenden. So kann die Laufzeit bei vielen Programmen signifikant verringert werden.

## 2.4 Grundlagen der Parallelverarbeitung

Parallelverarbeitung bezeichnet eine Technik, bei der man die Berechnung des Codes nicht nur auf einem Prozessor durchführt, sondern mehrere Prozessoren gleichzeitig mit Teilen der Berechnung beauftragt. Das Ziel ist, die Rechenzeit zu mindern. Aufgrund des hohen Performanzanspruchs bei Echtzeit-Anwendungen kommt Parallelverarbeitung dabei oft zum Einsatz [2, 8, 18]. Man spricht auch von Parallelisierung oder Multiprocessing [3].

Herausforderung bei diesem Vorgang ist einerseits die Zuweisung der Aufgaben an die verfügbaren Prozessoren (englisch: Task Scheduling) und andererseits das Abrufen und Bearbeiten gemeinsamer Ressourcen durch die Prozesse [17, 28].

Die Ray Bibliothek in Python stellt Funktionen zur Verfügung, um Aufgaben innerhalb einer Anwendung sinnvoll zu skalieren. Sie kümmert sich komplett um das Blockieren und Freigeben von geteilten Ressourcen. Palach widmete ein ganzes Buch der Entwicklung paralleler Systeme in Python [25]. Darin wird die Komplexität des Vorgangs deutlich.



# 3 Methodenentwicklung

Dieses Kapitel behandelt die vorgenommenen Schritte zur Erstellung des Tracking-Systems im Detail. Zu Beginn werden getroffene Annahmen geklärt, die künftigen Nutzern des Systems als Hilfestellung dienen sollen. Außerdem wird die Vorbereitung des Trackings erläutert, welche die Kalibrierung der Kamera und die korrekte Darstellung der Arena beinhaltet. Anschließend werden die Methoden zur Lokalisierung der Roboter und zur Berechnung ihrer Orientierungen präsentiert. Im Anschluss werden die beiden entwickelten Barcode-Designs vorgestellt und ihre Dekodierung über das Kamerabild näher betrachtet.

Nach der Extraktion aller wichtigen Informationen, behandelt das Kapitel die Aufbereitung der Ergebnisse durch Anzeigen und Protokollieren. Final werden Methoden zur Analyse und zur Verbesserung der Laufzeit des Programms dargelegt.

## 3.1 Getroffene Annahmen

Zur Erstellung des Tracking-Systems musste eine Reihe von Annahmen über die Kamera, die Arena und die Roboter getroffen werden. Diese Annahmen sind notwendig, um beispielsweise die Arena-Ecken oder die Roboter eindeutig zu identifizieren. Sie werden in folgender Liste zusammengefasst:

1. die gesamte Arena inklusive aller Ecken und Banden ist auf dem Kamerabild erkennbar,
2. der Kamerafokus ist so eingestellt, dass alle Roboter scharf zu sehen sind,
3. die Bildauflösung erfolgt in 4k,

## KAPITEL 3. METHODENENTWICKLUNG

4. die Belichtung ist so aufgestellt, dass keine Spiegelungen auf den Roboter-Bildschirmen oder den Banden auftreten,
5. in allen vier Arena-Ecken befinden sich rechtwinklige Dreiecke, deren Größe sich nicht ändert,
6. alle vier Dreiecke sind zum Zeitpunkt der Vorbereitung des Bildes erkennbar und nicht verdeckt,
7. der Helligkeitswert der Bande ist bedeutend kleiner als der des Arena-Hintergrundes,
8. die Arena ist rechtwinklig,
9. die kurzen Seiten der Arena haben die gleiche Länge,
10. die langen Seiten der Arena haben die gleiche Länge,
11. die kurzen Seiten der Arena sind halb so lang wie die langen Seiten,
12. die Roboter befinden sich ausschließlich auf der Arena,
13. jeder Roboter zeigt einen Barcode entsprechend der entwickelten Barcode-Designs auf seinem Bildschirm an,
14. die Größe der Roboter-Oberfläche und die des Roboter-Bildschirms ändern sich nicht,
15. die Roboterbildschirme sind so angebracht, dass der gelbe Bildschirmbereich in Laufrichtung des Roboters zeigt,
16. die Roboterbildschirme leuchten ausreichend hell,
17. es befinden sich keine weiteren Objekte mit den gleichen Farbtönen wie die des Roboterbildschirms auf der Arena.

## 3.2 Kamerakalibrierung

Zur Aufnahme der Arena wurde eine Kamera mit 4K Auflösung (4.096 x 2.160 Bildpunkte) mittig über der Arena befestigt. Wie in Abschnitt 2.1.1 beschrieben, wurde unsere Kamera mithilfe einer Kalibrierungsmatrix eingestellt. Dafür wurde die Kalibrierungsmatrix an einer Wand befestigt und ein Video von dieser gemacht, während die Kamera im Raum bewegt wurde. Wichtig ist, dass die Kalibrierungsmatrix in jedem Bereich des Bildes mindestens einmal zu sehen ist. Die Einzelbilder dieses Videos (insgesamt 67) wurden nun verwendet, um die Kameraeigenschaften mithilfe der OpenCV-Funktion *calibrateCamera()* zu ermitteln. Diese Eigenschaften werden der Funktion *undistort()* übermittelt, um das eingegebene Bild zu entzerrn.

Abbildung 3.1 ist eine originale Aufnahme der Arena ohne jegliche Bearbeitung. Abbildung 3.2 zeigt die gleiche Aufnahme nach der Entzerrung, deutlich wird dies vor Allem an den Ecken. Dort verlaufen die Arenagrenzen im Originalbild nicht exakt gerade, was bei dem entzerrten Bild fast perfekt gegeben ist. Nicht jeder Arenapunkt ist gleich weit von der Kamera entfernt. Mittige Punkte sind näher, Punkte nahe der Bande sind weiter weg. Auf dem Bild soll die Arena so dargestellt werden, als wären alle Punkte gleich weit von der Kamera entfernt. Dies gewährleistet, dass die berechneten Standorte der Roboter mit den physischen übereinstimmen.

## 3.3 Exakte Darstellung der Arena

Um die Arena korrekt auf dem Bildschirm darzustellen, müssen die vier Arena-Ecken identifiziert werden. Anhand dieser Ecken wird das Bild durch Homographie genau auf die Arena-Grenzen projiziert, siehe Abschnitt 2.1.2. Die Arena-Ecken sollen mit den Ecken des Bildes übereinstimmen. So wird eventuelle Rotation vermieden und eine bessere Darstellung gewährleistet.

## KAPITEL 3. METHODENENTWICKLUNG

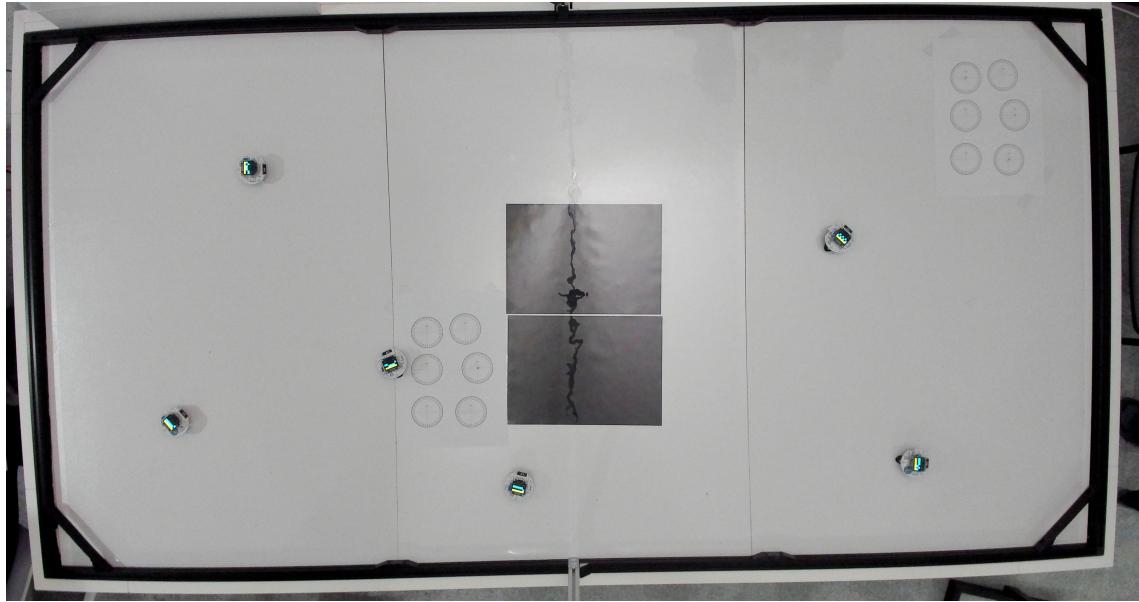


ABBILDUNG 3.1: Originalaufnahme der Arena.

### 3.3.1 Lokalisierung der Arena-Ecken

Die Arena-Ecken werden über Konturen im Bild ausfindig gemacht. OpenCV bietet hierfür die Funktion *findContours()* an, welche das Bild in schwarz-weiß entgegennimmt. Für diese Überführung des Farbraums wird das Bild zunächst binarisiert. Um verschiedenen Helligkeitsstufen innerhalb des Bildes gerecht zu werden, wurde hier eine adaptive Binarisierung mit der OpenCV-Funktion *adaptiveThreshold()* vorgenommen. Diese Funktion betrachtet für jeden Bildpunkt seine Nachbarschaft, also naheliegende Bildpunkte. Wie weit diese Nachbarschaft reicht, muss der Funktion mitgeteilt werden. In Versuchen hat sich ergeben, dass eine Nachbarschaftsgröße von 201 gute Ergebnisse für die binarisierten Bilder liefert. Die Nachbarschaft entspricht einem Quadrat mit einer Seitenlänge von 201 Bildpunkten und dem zu betrachtenden Bildpunkt in der Mitte. Wenn der Grauwert des Bildpunkts größer ist, als der mittlere Grauwert seiner Nachbarschaft, dann wird der Bildpunkt im binarisierten Bild schwarz. Ist dies nicht der Fall, dann wird er weiß eingefärbt.

In diesem binarisierten Bild können nun Konturen erkannt werden, also zusammenhängende Kanten. Alle gefundenen Konturen werden von der Funktion in einer Liste zusammengestellt. Aus dieser Liste von Konturen sollen nun die Konturen der Ecken

### 3.3. EXAKTE DARSTELLUNG DER ARENA

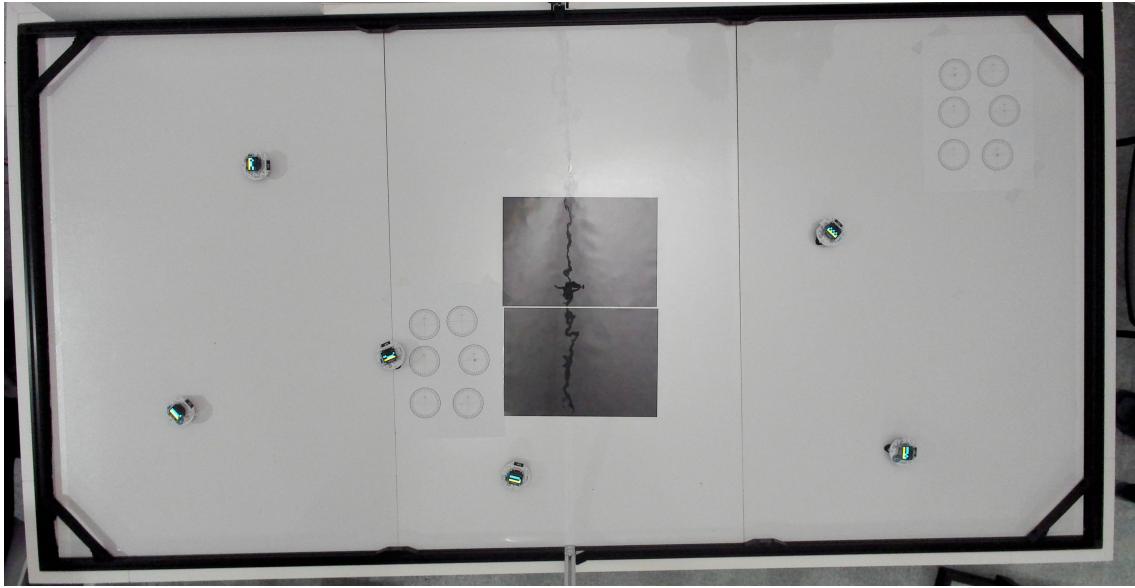


ABBILDUNG 3.2: Entzerrte Aufnahme der Arena.

gefiltert werden. Folgende Bedingungen werden an die Konturen gestellt, um sie als potenzielle Arena-Ecken anzunehmen:

1. Konturenlänge liegt zwischen 400 und 1000 Bildpunkten,
2. Konturenform hat drei Ecken,
3. Abstand der Ecken auf der linken Seite entspricht Abstand der Ecken der rechten Seite,
4. Abstand der Ecken der kurzen Seiten entspricht dem halben Abstand der Ecken der langen Seiten.

Die erste Bedingung an die Ecken ist ihre Größe. Anhand der Funktion *arcLength()* von OpenCV lässt sich die Länge der Kontur bestimmen. Ist diese zwischen 400 und 1000 Bildpunkten handelt es sich potenziell um eine Arena-Ecke. Eine weitere Bedingung ist die dreieckige Form. Wie auf den Abbildungen 3.1 und 3.2 zu sehen, ist in jeder Ecke eine zusätzliche Bande, die ein kleines Dreick entstehen lässt. Von daher kann man alle Konturen, die kein Dreieck bilden, vernachlässigen.

Es gibt zwei finale Bedingungen über die Beziehungen der Ecken zueinander: die Ecken auf der linken Seite haben in etwa die gleiche Entfernung voneinander wie die

### KAPITEL 3. METHODENENTWICKLUNG

beiden Ecken der rechten Seite, und die kürzeren Seiten sind circa halb so lang wie die beiden längeren Seiten. Zur Überprüfung dieser Bedingungen werden die äußeren Punkte der übrigen Konturen in vier Kandidatenlisten sortiert - jeweils eine für jede Ecke. Konturen, die auf der x-Achse zwischen dem Beginn, also null, und einem Viertel der Arena-Breite und in der oberen Hälfte der y-Achse liegen, sind Kandidaten für die linke obere Ecke. Liegen sie in der unteren Hälfte sind sie Kandidaten für die linke untere Ecke. Analog muss eine Kontur auf der x-Achse zwischen dem drei Viertel und der vollen Arena-Breite liegen, um potenziell eine Ecke auf der rechten Seite zu sein. Wie zuvor, wird die Kontur in die Kandidatenliste der oberen rechten Ecke aufgenommen, falls sie in der oberen Bildhälfte liegt. Liegt die Kontur in der unteren Bildhälfte wird sie in die Liste für die untere rechte Ecke eingereiht. Anhand dieser Kandidatenlisten lässt sich prüfen, bei welcher Kombination aus potenziellen Eckpunkten die zuvor beschriebenen Bedingungen erfüllt werden. Die Kandidaten, die die Bedingungen erfüllen, werden als Arena-Ecken angenommen.

Diese Prozedur wird einmal pro Experiment beim ersten Bild durchgeführt. Die ermittelten Positionen der Ecken werden für alle weiteren Bilder genutzt. Es ist bei jedem Experiment erneut notwendig, da sich die Position der Kamera gegebenenfalls leicht verändert.

Abbildung 3.3 zeigt die gleiche Arena-Aufnahme wie Abbildung 3.2 und enthält zusätzliche Informationen, die zur Erkennung der Arena-Ecken genutzt werden. In Hellblau werden alle Konturen gekennzeichnet, die der vorgegebenen Länge entsprechen. Von diesen Konturen werden nur Dreiecke weiter betrachtet. Die grünen Linien teilen das Bild in fünf Bereiche auf. Im mittleren Bereich können keine Ecken liegen. Dreieckige Konturen des oberen linken Bereichs sind Kandidaten für die obere linke Ecke, gleiches gilt für die restlichen drei Bereiche.

Die Längen der Verbindungslien, die in der Abbildung orange dargestellt sind, entscheiden über das Annehmen oder Ablehnen der Kandidaten. Die beiden kurzen Seiten müssen in etwa die gleiche Länge haben, hier liegt der Unterschied bei drei Bildpunkten. Maximal darf die Differenz bei 50 Bildpunkten liegen. Summiert haben die beiden kurzen Seiten eine Länge von 3.713 Bildpunkten. Der Mittelwert der Längen der beiden langen Seiten liegt bei 3.700 Bildpunkten. Die Differenz dieser beiden berechneten Werte liegt bei 13 Bildpunkten. Sie darf ebenfalls nicht größer als 50

### 3.3. EXAKTE DARSTELLUNG DER ARENA

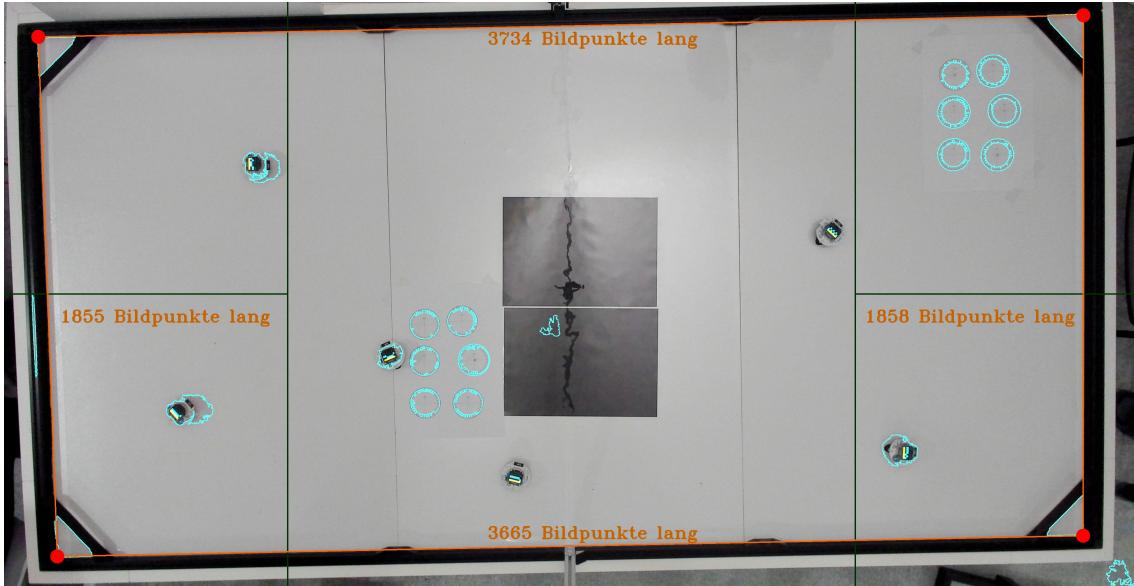


ABBILDUNG 3.3: Arena-Aufnahme mit eingezeichneten Informationen, die zur Erkennung der Arena-Ecken notwendig sind (hellblau: Konturen der vorgegebenen Länge, dunkelgrüne Linien: Begrenzungen für Bereiche, in denen die Ecken gesucht werden, rote Punkte: angenommene Ecken, orange: Verbindungslien zwischen den Ecken).

sein. Sind diese Bedingungen erfüllt, werden die Kandidaten als Ecken angenommen. Die angenommenen Ecken werden im Bild als rote Punkte dargestellt.

#### 3.3.2 Homographie

Um die Eckpunkte der Arena auf die Bild-Ecken zu projizieren, wird Homographie verwendet, siehe Abschnitt 2.1.2. Ein einfaches Zuschneiden auf die Arena-Ecken könnte wichtige Teile der Arena auslassen, falls es zu leichten Rotationen kam. Mit hilfe der Open-CV-Funktion `findHomography()` wird eine Matrix berechnet, die die Projektion der Arena-Ecken auf die Bild-Ecken beschreibt. Diese Matrix dient als Eingabe für die Funktion `warpPerspective()`, die die Perspektive des Bildes transformiert. Die Arena-Ecken wurden jeweils 50 Bildpunkte auf der x- als auch auf der y-Achse nach außen verschoben, um die Banden im Bild zu behalten und später keine Roboter auszuschließen. Das Resultat gibt Abbildung 3.4 wider.

## KAPITEL 3. METHODENENTWICKLUNG

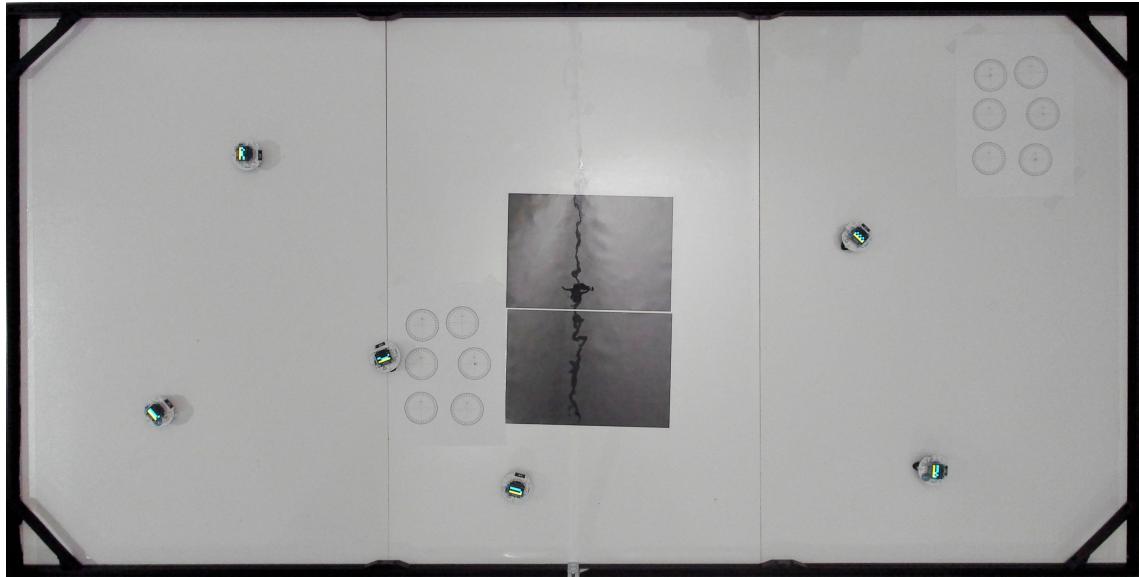


ABBILDUNG 3.4: Arena-Aufnahme nach der Homographie.

Die Homographie wird für jedes Einzelbild anhand der gefundenen Arena-Ecken des ersten Bildes durchgeführt.

## 3.4 Lokalisierung der Roboter

Nachdem das Bild fertig vorverarbeitet ist, folgt die Umsetzung des ersten Kriteriums des Trackings: dem Auffinden der Roboter. Diese Aufgabe ähnelt der Lokalisierung der Arena-Ecken, siehe Abschnitt 3.3.1. Entsprechend nahe kommt auch der Lösungsansatz. Wie zuvor werden Konturen auf dem binarisierten Bild gesucht und dann Bedingungen an diese Konturen gestellt. Die hier auftretende Herausforderung, die bei den Arena-Ecken nicht gegeben war, ist die Handhabung von Überschneidungen. Diese treten durch mehrere gefundene Konturen an einer Stelle auf, unter Anderem aufgrund von Schatten.

Final werden die Bildausschnitte, die Roboter beinhalten, festgelegt, ausgeschnitten und an kommende Funktionen weitergegeben. Abbildung 3.5 zeigt, wo Konturen gefunden wurden (gelb), welche dieser Konturen die Bedingungen erfüllen, aber auf-

### 3.4. LOKALISIERUNG DER ROBOTER

grund von Überschneidungen abgelehnt werden (blau), und welche der Kandidaten als Roboter angenommen werden (rot).

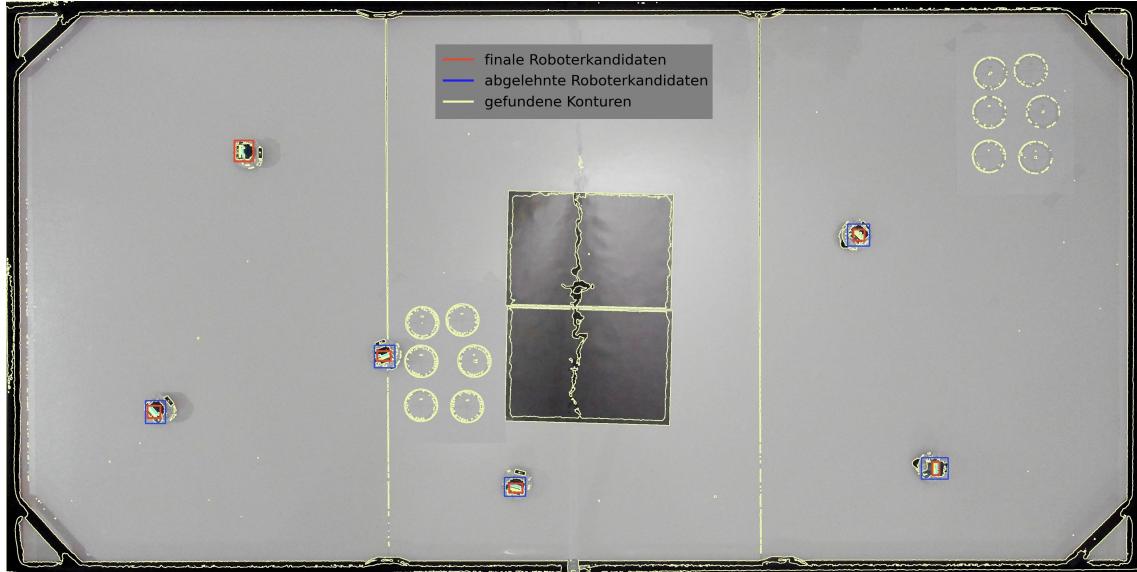


ABBILDUNG 3.5: Arena-Aufnahme mit allen Konturen (gelb), aufgrund von Überschneidungen abgelehnten Roboterkandidaten (blau) und finalen Roboterkandidaten (rot).

#### 3.4.1 Bedingungen zur Roboteridentifizierung

Folgende Bedingungen werden an die gefundenen Konturen gestellt:

1. Konturenfläche liegt zwischen 300 und 1500 Bildflächeneinheiten,
2. Kontur beinhaltet ausreichend Bildpunkte in vorgegebenem blauen Farbbereich,
3. Kontur beinhaltet ausreichend Bildpunkte in vorgegebenem gelben Farbbereich.

Als erste Bedingung dient die Größe der Kontur: in diesem Fall nicht die Länge sondern die eingeschlossene Fläche, die mit `contourArea()` von OpenCV berechnet werden kann. Sie sollte weder kleiner als 300 noch größer als 15000 Bildflächeneinheiten sein. Der Bereich ist sehr weit gefasst, um keine Konturen auszuschließen, die

## KAPITEL 3. METHODENENTWICKLUNG

lediglich den Roboterbildschirm gefunden haben, noch jene, die den Roboter samt seinem Schatten identifiziert haben. Diese erste Bedingung soll jene Konturen entfernen, die offensichtlich keine Roboter sein können.

Die zweite Bedingung überprüft, ob es einen gelben und einen blauen Bereich innerhalb der Kontur gibt und ob dieser groß genug ist. Für diese Aufgabe sind mehrere Schritte notwendig. Um nur den Bereich des Bildes zu betrachten, in dem die Kontur liegt, wird eine so genannte Bounding Box um die Kontur gelegt anhand der Open-CV-Funktion *boundingRect()*. Die Bounding Box ist das kleinstmögliche Rechteck, welches alle Punkte der Kontur beinhaltet. Sie wird aus dem Gesamtbild ausgeschnitten, um die nächsten Verarbeitungsschritte durchzuführen. Der Bildausschnitt wird zuerst in das Hue-Saturation-Value (HSV) Farbmodell überführt. Sowohl vom Blau- als auch vom Gelbton der Roboterbildschirme wurden vorher die Farbwerte analysiert, um einen Farbbereich festzulegen. Dafür eignet sich das HSV-Modell sehr gut, da die Saturation und die Helligkeit (value) getrennt von der Farbe selbst dargestellt werden. So ließ sich festlegen, dass der Hue-Wert des Gelbs zwischen 25 und 35 liegt und der des Blaus zwischen 85 und 95. In beiden Fällen werden jegliche Saturationswerte über 20 zugelassen. Der Helligkeitswert muss sowohl bei Blau als auch bei Gelb sehr hoch sein, mindestens 180. Im standardmäßigen RGB-Modell ist die Identifizierung eines solchen Farbbereichs wesentlich komplexer.

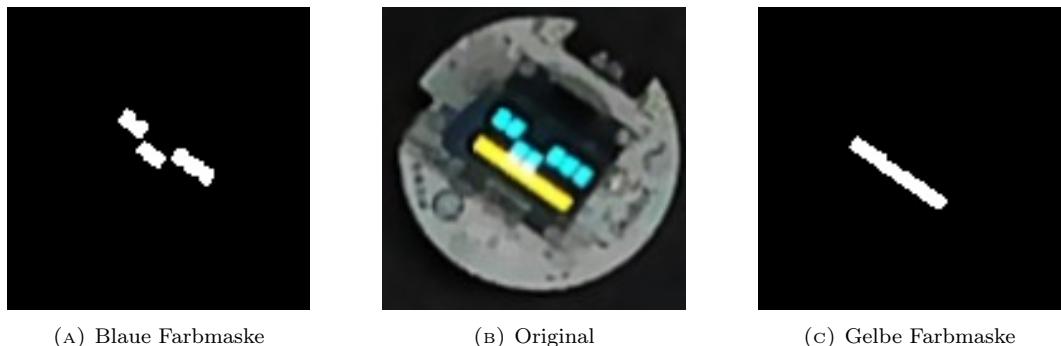


ABBILDUNG 3.6: Gelbe und Blaue Farbmasken im Vergleich zum Originalbildausschnitt

Anhand der *inRange()*-Funktion lässt sich eine Maske erstellen, die Bildpunkte weiß darstellt, in denen der vorgegebene Farbbereich eingehalten wird. Alle anderen Bildpunkte werden schwarz dargestellt. Die Maske wird sowohl für den blauen als auch

### 3.4. LOKALISIERUNG DER ROBOTER

für den gelben Farbbereich ermittelt. Abbildung 3.6 zeigt ein Beispiel für die beiden Farbmasken im Vergleich zum originalen Bildausschnitt des Roboters. Auf diesen beiden Masken werden wiederum die Konturen gesucht. Gibt es keine Konturen, ist die gesamte Maske schwarz. Das heißt, dass es die Farbe auf dem Bild nicht gibt. Somit kann es sich bei diesem Bildausschnitt nicht um einen Roboter handeln. Falls Konturen gefunden werden, wird die längste ermittelt. Nur wenn die längsten Konturen beider Farbbereiche mehr als sieben Längeneinheiten aufweisen, wird der Bildausschnitt als Roboter angenommen.

#### 3.4.2 Handhabung von Überschneidungen

Wenn die Roboterkandidaten, wie in Abschnitt 3.4.1 beschrieben, ermittelt werden, kommt es oft dazu, dass statt einem Kandidaten mehrere an der gleichen Stelle identifiziert werden. Beispielsweise kann es zu Überschneidungen kommen, wenn der Schatten eines Roboters gefunden wurde. In dem Fall, dass zwei Roboter direkt aneinander stehen, kann es vorkommen, dass unter Anderem eine große Kontur, die beide inkludiert, gefunden wird. Um diese Fehl- und Doppelerkennungen zu vermeiden, müssen Überschneidungen von Roboterkandidaten sinnvoll gehandelt werden. Jegliche Kombinationen der Roboterkandidaten werden anhand ihrer Bounding Box, also dem einschließenden Rechteck, geprüft. Zuerst wird begutachtet, ob eines der beiden Rechtecke innerhalb des anderen liegt. Ist dies der Fall, wird das äußere der beiden aus der Kandidatenliste entfernt. Somit werden doppelte Erkennungen weitgehend ausgeschlossen. Wenn sich zwei Rechtecke überschneiden, muss überprüft werden, wie groß die Schnittfläche ist. Bei kleinen Schnittflächen kann es sich um benachbarte Roboter handeln, bei größeren muss einer der beiden Kandidaten entfernt werden.

Der Codeabschnitt 3.1 zeigt, wie die Schnittfläche zweier Rechtecke ermittelt werden kann, siehe Abbildung 3.7. In der Abbildung entspricht Punkt 1 den Koordinaten  $x_3$  und  $y_1$ , während Punkt 2 bei  $x_2$  und  $y_4$  liegt. Die Fläche des eingeschlossenen Rechtecks wird über seine Seitenlängen bestimmt. Ist eine der Seitenlängen negativ, gibt es keine Überschneidung und somit ist die Schnittfläche null.

### KAPITEL 3. METHODENENTWICKLUNG

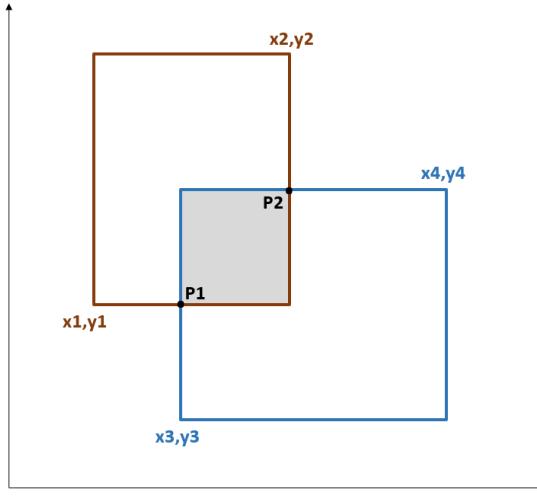


ABBILDUNG 3.7: Beispiel für eine Überschneidung von zwei Rechtecken.

```
def calculate_intersection_area(x1, y1, x2, y2, x3, y3, x4, y4):
    p1_x = max(x1, x3)
    p1_y = max(y1, y3)
    p2_x = min(x2, x4)
    p2_y = min(y2, y4)
    intersection_area = max(p2_x - p1_x, 0) * max(p2_y - p1_y, 0)
return intersection_area
```

LISTING 3.1: Berechnung Schnittfläche.

Ist die Schnittfläche größer als 500 Flächeneinheiten so muss eines der beiden Rechtecke eliminiert werden. Die Entscheidung darüber, welches der beiden Rechtecke überflüssig ist, wird über die Länge der farbigen Konturen im Bildausschnitt getroffen, die bereits zum Ende von Abschnitt 3.4.1 ermittelt wurden. Behalten wird der Kandidat bei dem die Summe der blauen und der gelben Konturlänge größer ist, was heißt, dass in diesem Bildausschnitt mehr von dem Roboterbildschirm zu sehen ist. Für den Fall, dass der Roboter und sein Schatten separat erkannt wurden, würde der Schatten durch dieses Verfahren ausgesondert werden. Wenn bei beiden Bildausschnitten die Summe der Konturlängen gleich ist, wird überprüft ob nur die gelbe Konturlänge bei einem der beiden größer ist. Der mit der kürzeren wird entsprechend entfernt. Sollte dies noch immer nicht zu einer Entscheidung führen, wird

### 3.5. ORIENTIERUNG DER ROBOTER

das Rechteck mit der größeren Fläche aussortiert. Dieses Vorgehen resultiert aus der Beobachtung, dass kleinere Rechtecke um die Bildschirme herum oft präziser in Bezug auf die Position sind. In diesem Fall ist die Entscheidung, welches Rechteck entfernt wird, von geringer Bedeutung.

#### 3.4.3 Festlegen der Roboterbildausschnitte

Sobald alle Roboter identifiziert und Duplikate entfernt wurden, wird das Zentrum der gelben Kontur vorerst als Mittelpunkt des Roboters festgelegt. Dieser kann mithilfe der Bildmomente berechnet werden, OpenCV stellt für die Berechnung die Funktion *moments()* zur Verfügung. Bildmomente beschreiben ein bestimmtes gewichtetes Mittel der Intensitäten der Bildpunkte.

Falls der Mittelpunkt so nicht gefunden werden kann, wird der Mittelpunkt der Bounding Box um die gelbe Kontur genutzt. Dieses Verfahren wurde dem Ermitteln des Zentrums des Rechtecks, das den Roboter einschließt, vorgezogen, um ein robusteres Ergebnis zu erzielen. Die Rechtecke können von Bild zu Bild variieren, doch der gelbe Balken des Roboterbildschirms ist beständig.

Für die weitere Verarbeitung sind Bildausschnitte der einzelnen Roboter von Nöten. Dafür wird ein 120 Bildpunkte hoher und 120 Bildpunkte breiter Ausschnitt ausgehend vom Zentrum des Roboters gewählt. Für den Fall, dass der Roboter am Arenarand steht und daher die Bildausschnittsgröße nicht zustande kommen kann, wird das Bild nur bis zum Rand ausgeschnitten. Die folgende Bestimmung der Orientierungen und Roboternummern erfolgt über diese Bildausschnitte der einzelnen Roboter.

## 3.5 Orientierung der Roboter

Im folgenden werden für jeden Roboter die gleichen Verfahren angewandt. Daher werden die Roboter unabhängig voneinander betrachtet. Die Orientierung des Roboters ist definiert über seinen Bildschirm. Der Bildschirm ist auf dem Roboter so angebracht, dass der gelbe Balken immer die Laufrichtung angibt. Wenn der gelbe

## KAPITEL 3. METHODENENTWICKLUNG

Balken exakt parallel zum oberen Bildrand steht und näher am oberen Bildrand als der blaue Bildschirmteil liegt, dann entspricht seine Orientierung  $0^\circ$ . Durch Drehung des Roboters gegen den Uhrzeigersinn nimmt der Orientierungswinkel zu, bis maximal  $359^\circ$ . Abbildung 3.8 zeigt einen Dezibot und den entsprechenden Winkelkreis. Der abgebildete Dezibot hat eine Orientierung von  $144^\circ$ . Zur Bestimmung dieser Orientierung wird der gelbe Balken approximiert und der Winkel des Balkens relativ zur x-Achse berechnet.

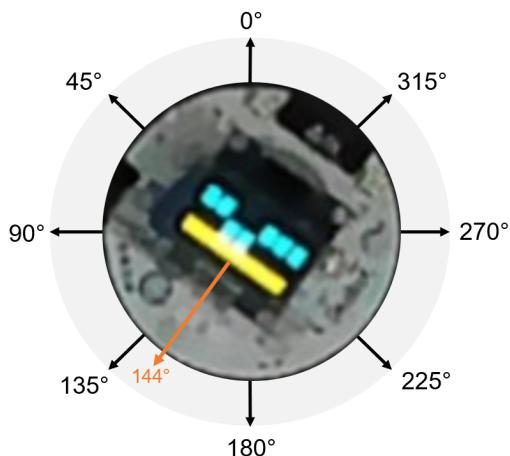


ABBILDUNG 3.8: Dezibot mit Winkelkreis.

### 3.5.1 Approximierung des gelben Balkens

Der gelbe Balken auf dem Bildschirm soll durch eine Linie approximiert werden, die möglichst mittig durch den Balken verläuft. Das bereits in Abschnitt 3.4.1 beschriebene Verfahren zum Finden der längsten Farbkontur durch eine Farbmaske wird hier erneut für die Farbe gelb verwendet. Um diese Farbkontur wird eine Bounding Box gelegt und mithilfe der OpenCV-Funktion `boxPoints()` werden alle Eckpunkte dieses Rechtecks ermittelt.

Um die mittlere Linie des Rechtecks zu finden, wird geprüft, welche der Eckpunkte an den kürzeren Kanten benachbart sind. Die beiden Mittelpunkte der kürzeren Kanten werden als Start- und Endpunkt der approximierten gelben Linie angenommen.

### 3.5. ORIENTIERUNG DER ROBOTER

Abbildung 3.9 zeigt einen Dezibot, auf welchem die Bounding Box um den gelben Balken in rot und die approximierte Mittellinie des gelben Balkens in schwarz eingezeichnet ist.

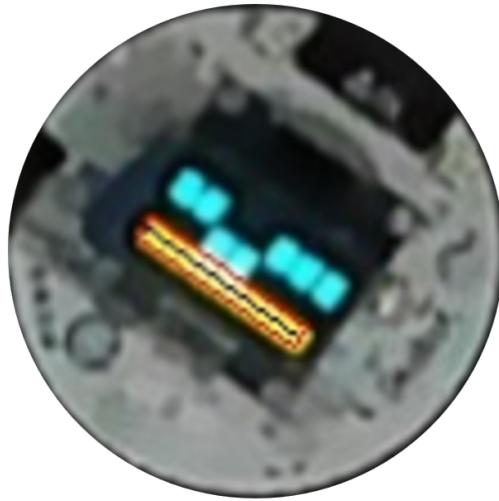


ABBILDUNG 3.9: Dezibot mit Bounding Box um den gelben Balken (rot) und approximierter Mittellinie (schwarz).

#### 3.5.2 Berechnung des Orientierungswinkels

Die Orientierung des Roboters kann über den Winkel zwischen der gelben Mittellinie und der x-Achse des Bildes ermittelt werden. Diese Achse entspricht nach der Homographie exakt der Orientierung der Arena. Die x-Achse wird durch den Vektor  $[1,0]$  repräsentiert. Der Vektor, der der approximierten gelben Mittellinie entspricht, wird zuerst auf die Länge eins normiert. Dies passiert in der Funktion `unit_vector()` im Codebeispiel 3.2, indem der Vektor durch seine Norm, also seiner Länge, geteilt wird.

Mithilfe der Numpy-Bibliothek in Python wird zum einen das Kreuzprodukt der beiden Vektoren gebildet und zum anderen die Determinante der zusammengefassten Matrix ermittelt. Das Kreuxprodukt und die Determinante werden in die Numpy-Funktion `atan2()`, der inversen Tangente, eingegeben, welche den Winkel im Bogenmaß zurückgibt. Dieser wird schließlich noch mit der Funktion `degrees()` in Grad umgewandelt.

## KAPITEL 3. METHODENENTWICKLUNG

Der Code in Listing 3.2 zeigt die fertige Funktion `angle_to_x_axis()`. Als Eingabe dienen die Koordinaten des Start- und Endpunktes der gelben Mittellinie. Die inverse Tangente stellt sicher, dass das Vorzeichen des Winkels bewahrt wird. Das ist von Bedeutung, da der Winkel zwischen  $0^\circ$  und  $359^\circ$  liegen soll. Andernfalls würde immer nur der kleinstmögliche Winkel ermittelt werden.

```
def unit_vector( vector ):
    if len( vector ) == 0: return 0
    return vector / np.linalg.norm( vector )

def angle_to_x_axis( x1, y1, x2, y2 ):
    v0 = [1, 0]
    v1 = unit_vector([x1 - x2, y1 - y2])
    determinant = np.linalg.det([v0, v1])
    dot_product = np.dot(v0, v1)
    angle = np.math.atan2(determinant, dot_product)
    return np.degrees(angle)
```

LISTING 3.2: Berechnung des Winkels der gelben Linie relativ zur x-Achse.

Desweiteren muss festgestellt werden, ob die gelbe Mittellinie über oder unter dem blauen Bildschirmteil liegt. Der Winkel relativ zur x-Achse gibt darüber keine Auskunft. Sowohl für einen Winkel von  $90^\circ$  als auch von  $270^\circ$  zwischen gelben Balken und x-Achse würden  $90^\circ$  berechnet werden. Daher muss der Winkel gegebenenfalls um  $180^\circ$  geändert werden, je nach dem, in welchem Verhältnis der gelbe und der blaue Bildschirmteil stehen.

Für diese Analyse werden die Mittelpunkte der längsten blauen und der längsten gelben Farbkontur ermittelt. Der Vektor vom gelben Mittelpunkt zum blauen Mittelpunkt wird um den ermittelten Winkel rotiert. Dafür wird eine Rotationsmatrix aus dem Winkel im Bogenmaß, genannt  $\theta$  (Theta), wie folgt berechnet:

$$\text{Rotationsmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix}$$

### 3.6. DESIGN DER BARCODES

Aus dieser Rotationsmatrix und dem Vektor zwischen den Farbmittelpunkten wird das Kreuzprodukt gebildet, um die Rotation des Vektors durchzuführen. Wenn die y-Koordinate des rotierten Vektors negativ ist, man also in negative y-Richtung von gelb nach blau geht, dann muss der Winkel um  $180^\circ$  erweitert werden. Zu beachten ist hier, dass der Koordinatenursprung in der linken oberen Ecke des Bildes liegt und die positive y-Achse nach unten gerichtet ist.

Final werden positive Orientierungswinkel von  $360^\circ$  abgezogen, um sie gegen den Uhrzeigersinn zu richten und von Orientierungswinkeln mit negativen Vorzeichen wird der Betrag gebildet. Mit diesem Vorgehen können die Orientierungen genau so ermittelt werden, wie in Abbildung 3.8 vorgegeben.

## 3.6 Design der Barcodes

Nachdem die Positionen und die Orientierungen der Roboter bestimmt sind, kann die Roboternummer identifiziert werden. Dafür wurden zwei Barcodes für den vorhandenen Roboterbildschirm entwickelt, der die Roboternummer kodiert. Der Bildschirm, welcher auf jedem Dezibot angebracht ist, hat eine Diagonale von drei Zentimetern mit  $128 \times 64$  Bildpunkten, die für den Barcode genutzt werden können. Der Bildschirm ist monochrom, das heißt, dass die Farben nicht geändert werden können. Der obere Teil ist immer gelb und der untere Teil immer blau. Der gelbe Teil leuchtet bei jedem Roboter dauerhaft, um die Orientierung bestimmen zu können. In dem blauen Bildschirmteil wird die Roboternummer kodiert.

Die in Abschnitt 2.2 vorgestellten Standards kodieren weitaus mehr Information als in diesem Anwendungsfall von Nöten ist. Aufgrund der geringen Abmessungen kann nicht zu viel Information kodiert werden, andernfalls könnte die Dekodierung fehlerhaft werden.

Auch die Größe der Arena ist begrenzt und somit die maximale Anzahl an Robotern, die sich gleichzeitig darauf befinden kann. Nach gründlicher Überlegung, wurde der Entschluss gefasst, dass die Roboternummer mit sieben Bits präsentiert werden soll. Somit können maximal 126 Dezibots gleichzeitig auf der Arena sein. Aktuell gibt es circa 30 Roboter vor Ort, weitere 40 wurden bestellt. Es ist unrealistisch, dass

## KAPITEL 3. METHODENENTWICKLUNG

auf dieser Arena eines Tages mehr als 126 Roboter gleichzeitig in einem Experiment involviert sind.

### 3.6.1 Barcode-Design 1

Der erste Barcode wurde wie folgt entworfen. Der blaue Bildschirmteil wurde in eine obere und eine untere Zeile aufgeteilt. Beide Zeilen bestehen aus sieben Spalten. Die obere Zeile präsentiert die Binärschreibweise der Roboternummer. Dort wo eine eins steht wird der Block beleuchtet, bei einer null bleibt er dunkel. Die untere Zeile invertiert die obere Zeile. Das heißt, dass in jeder Spalte, in der der Block in der oberen Spalte leuchtet, der Block in der unteren Zeile nicht leuchtet und andersherum. Diese Invertierung stellt einen zusätzlichen Absicherung durch Wiederholung des Codes dar.

Abbildung 3.10 zeigt den Barcode auf dem Bildschirm des Dezibots mit der Roboternummer elf. Die obere Zeile kodiert die Binärschreibweise der Roboternummer durch leuchtende Blöcke, die untere Zeile liefert die Inverse. Unter dem Bildschirm ist die Berechnung der Roboternummer aufgezeigt. Mit diesem Code ist es nicht möglich die Roboternummern 0 und 127 darzustellen, da in beiden Fällen eine der beiden Zeilen unbeleuchtet ist und somit das Ende des Bildschirms nicht fehlerfrei erfasst werden kann. Insgesamt können also 126 Roboternummern kodiert werden.

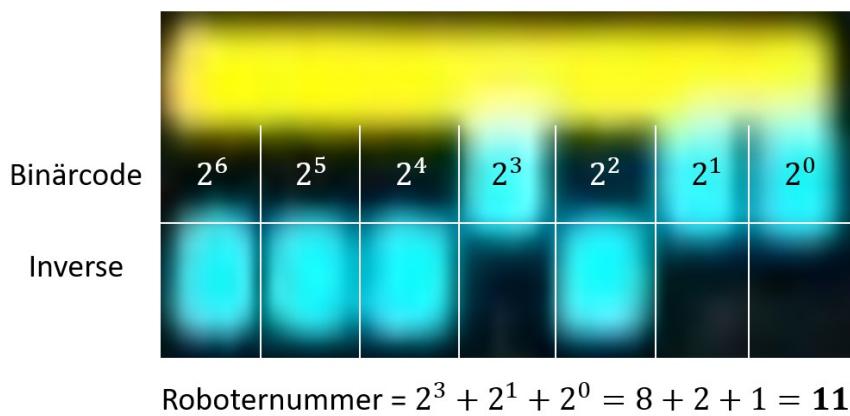


ABBILDUNG 3.10: Barcode mit Erläuterung zur Kodierung, Design 1.

### 3.6.2 Barcode-Design 2

Das zweite Barcode-Design wurde folgendermaßen konzipiert. Nach wie vor muss der gelbe Bildschirmteil durchgehend leuchten, um die Orientierung festzustellen. Der blaue Bildschirmteil wurde wie beim 1. Design in sieben Spalten und zwei Zeilen aufgeteilt. Ebenfalls gleich bleibt, dass immer genau die Hälfte der Blöcke leuchten. Der einzige Unterschied liegt in der Bedeutung der Blöcke für die Roboternummer. In diesem Design wird die Binärschreibweise der Roboternummer nicht wiederholt, sondern auf die entstehenden Blöcke verteilt.

Immer zwei benachbarte Blöcke bilden ein Blockpaar. Die ersten sechs Blöcke von links der unteren Zeile geben Auskunft über die hinteren drei Stellen der Binärschreibweise der Roboternummer. Das Bit ist eins, wenn der rechte Block des Blockpaares leuchtet, sonst ist es null. Die Spalte ganz rechts beinhaltet die vierte Stelle der Binärschreibweise der Roboternummer. Leuchtet der obere Block, dann ist das Bit gesetzt. Die restlichen Blöcke der oberen Spalte repräsentieren die restlichen drei Bits der Binärschreibweise der Roboternummer. Das jeweilige Bit ist gesetzt, wenn der linke Block des Blockpaares leuchtet.

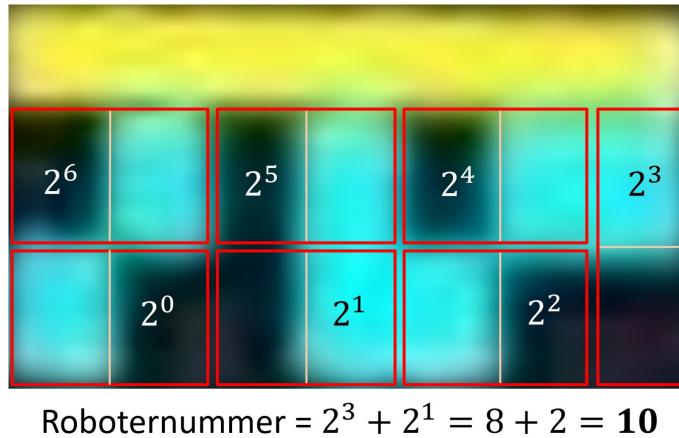


ABBILDUNG 3.11: Barcode mit Erläuterung zur Kodierung, Design 2.

Abbildung 3.11 zeigt ein Beispiel für dieses Barcode-Design auf einem Roboter-Bildschirm. Die roten Rahmen zeigen an, welche zwei Blöcke jeweils gemeinsam für ein Bit verantwortlich sind und somit ein Blockpaar bilden. Die weißen Linien inner-

## KAPITEL 3. METHODENENTWICKLUNG

halb der Blockpaare zeigen die Trennung der beiden Blöcke auf. Die Zweierpotenzen in den Blockpaaren zeigen an, für welches Bit dieses Blockpaar verantwortlich ist. Die Zweierpotenz ist in dem Block des Blockpaars eingezeichnet, der leuchten muss, damit das Bit gesetzt ist. Ist das Bit dieses Blockpaars gesetzt, also leuchtet der entsprechende Block, so wird diese Zweierpotenz zum Ergebnis hinzuaddiert. In diesem Beispiel sind das erste und das dritte Bit gesetzt, woraus sich die Roboternummer zehn ergibt. In Binärschreibweise ist sie in diesem Fall 0001010.

Dieses Barcode-Design bietet den Vorteil, dass die beiden Zeilen unabhängig voneinander sind und der Barcode insgesamt heterogener ist. Außerdem können gewisse Fehler identifiziert werden. Innerhalb eines Blockpaars kann immer nur einer der beiden Blöcke leuchten. Pro Zeile können nur drei beziehungsweise vier Blöcke leuchten. Es können niemals drei aufeinander folgende Blöcke gleichzeitig leuchten oder dunkel bleiben. Wird eine dieser Annahmen verletzt, so gab es einen Fehler in der Erkennung, den man entsprechend anzeigen kann. Ein weiterer Vorteil ist, dass mit diesem Barcode-Design auch die Roboternummern null und 127 verwendet werden können.

### 3.7 Dekodierung der Barcodes

Die entworfenen Barcodes werden auf die Roboterbildschirme geladen und sollen im folgenden Schritt anhand der Kameraaufnahme dekodiert werden. Die in Abschnitt 3.4 gefundenen Roboterausschnitte und die in Abschnitt 3.5 ermittelten Orientierungen werden in diesem Abschnitt erneut verwendet, um die Barcodes zu dekodieren. Die gelbe Linie wird im Winkel der Orientierung über den Bildschirm geschoben. So entstehen parallele Linien, die als Abtastlinien (englisch: scanlines) dienen. Auf diesen Linien wird die Helligkeit des Blaukanals des Bildausschnitts abgetastet. Die Helligkeitslisten für jede Abtastlinie können in einen sieben Bit langen Binärcode umgewandelt werden. Zuletzt erfolgt eine Abstimmung unter allen Abtastlinien.

### 3.7.1 Erstellung von Abtastlinien

Die Abtastlinien werden parallel zur gelben Linie über den Roboterbildschirm gelegt. Sie werden erstellt, indem die gelbe Linie durch Parallelverschiebung unter Berücksichtigung der Orientierung verlegt wird. Dafür werden sowohl Start- als auch Endpunkt der gelben Linie im Winkel der Orientierung verschoben. Es werden Abtastlinien mit einer Distanz zwischen drei und 20 Längeneinheiten zur gelben Linie erstellt. Die Verschiebung der Punkte erfolgt über Dreiecksbeziehungen. Nur bei offensichtlichen Fällen ( $0^\circ$ ,  $90^\circ$ ,  $180^\circ$  und  $270^\circ$ ) werden die neuen Punktkoordinaten direkt gebildet.

Mithilfe des Bresenham-Algorithmus und der gleichnamigen Python-Bibliothek werden alle Bildpunkte, die auf der Linien zwischen Start- und Endpunkt der Abtastlinie liegen, in einer Liste gespeichert. Für die entstandenen Listen wird überprüft, ob sie durch den blauen Teil des Bildschirms verlaufen. Für manche erstellte Linien kann es sein, dass der blaue Bildschirmbereich noch nicht begonnen oder schon aufgehört hat. Der Entfernungsbereich variiert bei den verschiedenen Roboterbildausschnitten. Ziel ist es, so viele Abtastlinien wie möglich über den blauen Bildschirmbereich zu legen, um später ein aussagekräftiges Ergebnis bei der Abstimmung zu erzielen.

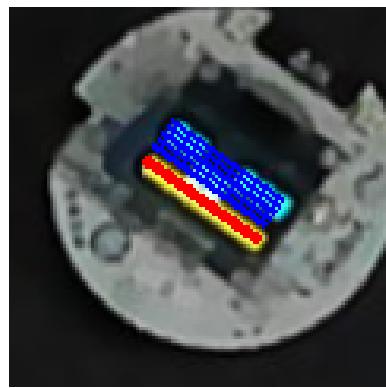


ABBILDUNG 3.12: Originalbild eines Dezibots mit der gelben Mittellinie (rot) und den erstellten Abtastlinien (blau).

Um sicherzustellen, dass die Linien über dem blauen Bildschirmbereich liegen, wird erneut die blaue Farbmaske des Roboterbildes genutzt, wie sie beispielhaft in Abbildung 3.6 zu sehen ist. Für jeden Bildpunkt der Abtastlinie wird auf dieser Farbmaske

## KAPITEL 3. METHODENENTWICKLUNG

überprüft, ob er weiß ist. Das würde bedeuten, dass der Bildpunkt im blauen Farbbereich liegt. Mindestens drei Bildpunkte der Linie müssen dieses Kriterium erfüllen, um die Linie als Abtastlinie anzunehmen. Drei Bildpunkte entsprechen ungefähr der Länge eines Blocks. Die rechte Seite der Abbildung 3.12 stellt den originalen Bildausschnitt eines Roboters dar, auf welchem die gefundene gelbe Mittellinie und die ermittelten Abtastlinien eingezeichnet sind.

### 3.7.2 Auslesen der Abtastlinien

Vom originalen Bildausschnitt des Roboters wird im Folgenden nur der Blau-Kanal betrachtet. Bei diesem sind die leuchtenden Blöcke besonders deutlich, wie man links in Abbildung 3.13 sieht.

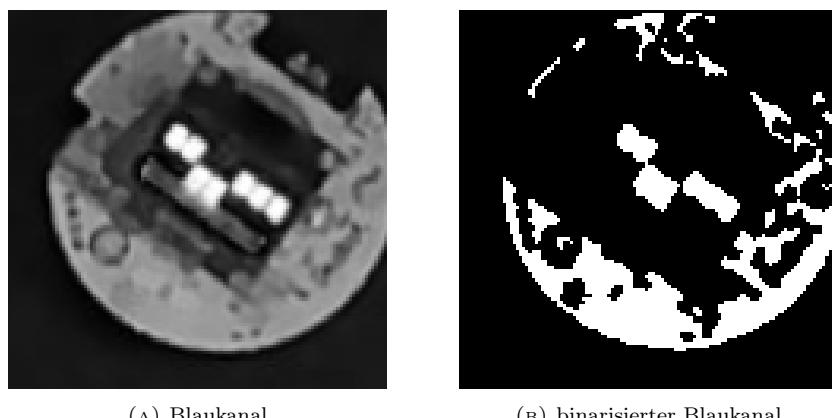


ABBILDUNG 3.13: Blaukanal und binarisierte Blaukanal eines Dezibot-Bildausschnitts mit dem mittleren Helligkeitswert des Bildschirms als Threshold für die Binarisierung.

Für jeden Bildpunkt einer jeden Abtastlinie wird der Helligkeitswert des Blau-Kanals geprüft und in einer Liste gespeichert. Aus der Summe der Helligkeiten und der Anzahl der Bildpunkte aller Abtastlinien wird ein mittlerer Helligkeitswert des blauen Bildschirmbereichs ermittelt. Dieser wird im folgenden als Schwellenwert (englisch: threshold) zur Binarisierung genutzt. Er ist besonders geeignet, da immer genau die Hälfte der Bildpunkte beleuchtet und die andere Hälfte dunkel sind. Dieser positive Effekt resultiert aus den Designs der Barcodes, bei denen exakt die Hälfte aller

### 3.7. DEKODIERUNG DER BARCODES

Blöcke leuchtet. Abbildung 3.13 zeigt auf der rechten Seite, wie der Blaukanal des Roboterbildes nach der Binarisierung aussieht. Die leuchtenden Blöcke sind deutlich von den restlichen unterscheidbar.

Für jede Abtastlinie wird die Sequenz der Helligkeitswerte auf dem Blaukanal durch Binarisierung in eine Sequenz von Nullen und Einsen umgewandelt. Für alle Bildpunkte jeder Abtastlinie wird geprüft, ob sein Helligkeitswert des Blaukanals über dem Schwellenwert liegt. Falls zutreffend, wird der Helligkeitswert durch eine Eins ersetzt. Andernfalls durch eine Null. Diese Bitsequenzen zeigen an, wo der Helligkeitswert über dem Schwellenwert liegt. Sie werden anschließend zur Dekodierung genutzt.

#### 3.7.3 Dekodierung jeder Abtastlinie

Im Folgenden wird jede Abtastlinie dekodiert. Das bedeutet, dass die Bitsequenzen, die durch die Binarisierung der Helligkeitswerte entstanden sind, in einen sieben-stelligen Binärkode übersetzt werden sollen. Da die Bitsequenzen in ihrer Länge aufgrund der Orientierung und Position des Roboters variieren, ist diese Aufgabe nicht trivial zu lösen.

Ein offensichtlicher Ansatz ist das Aufteilen der Bitsequenz in sieben Teile mit anschließender Analyse jedes Teilbereichs. Aufgrund der Heterogenität der Bildausschnitte ist nicht gegeben, dass die Abtastlinien bei jedem dieser Bilder exakt an den Rändern des Bildschirms beginnen und enden. Somit kommt es häufig zu falsch definierten Bereichen, die das Ergebnis verzerren.

Aufgrund dessen wurde ein verfeinerter Ansatz zur Dekodierung entwickelt. Jede Bitsequenz wird separat betrachtet. Der Vorgang wird für alle durchgeführt. Zuerst wird festgestellt, wie viele Bits der Sequenz auf ein Bit des Binärcodes fallen, indem die Länge der Bitsequenz durch die Länge des Binärcodes - also sieben - geteilt wird. Nacheinander werden alle Bits der Bitsequenz durchlaufen. Für jedes Bit wird überprüft, ob es das gleiche wie das vorherige ist. Wenn ja, dann wird ein Zähler inkrementiert. Ist dies nicht der Fall, so wird der Binärkode erweitert. Der Zähler mit zuvor gleich gesetzten Bits wird durch das Verhältnis von Sequenzlänge zu Binärcodelänge geteilt. Das Ergebnis gibt an, wie viele Bits dem finalen Binärkode

## KAPITEL 3. METHODENENTWICKLUNG

hinzugefügt werden müssen, das Minimum ist Eins. Es wird in einer Liste vermerkt, wie viele Bits der Sequenz für ein Bit des Binärcodes verantwortlich sind. Diese Information ist im Folgenden von Signifikanz, wenn es darum geht, den Binärcode auf die Länge sieben zu bringen.

Dieses Vorgehen erstreckt sich über die gesamte Sequenz, ausgenommen dem letzten Bit. Bei diesem werden nochmals Bits dem finalen Binärkode nach dem gleichen Schema zugefügt, vorausgesetzt der Zähler liegt bei mindestens Zwei.

Nach der Umwandlung in den Binärcode kann es sein, dass dessen Länge nicht sieben entspricht, je nach Länge und Ausbildung der Bitsequenz. Für diesen Fall wird die Information, wie viele Bits der Sequenz für wie viele Bits des Binärcodes verantwortlich sind, genutzt. Ist die finale Länge kürzer als sieben, so werden die Bits im Binärcode wiederholt, die durch die meisten Bits der Bitsequenz verursacht wurden. Liegt die Länge bei mehr als sieben, so werden die Bits des Binärcodes entfernt, die durch die wenigsten Bits der Sequenz entstanden sind. So ergibt sich für jede Abtastlinie ein siebenstelliger Binärcode.

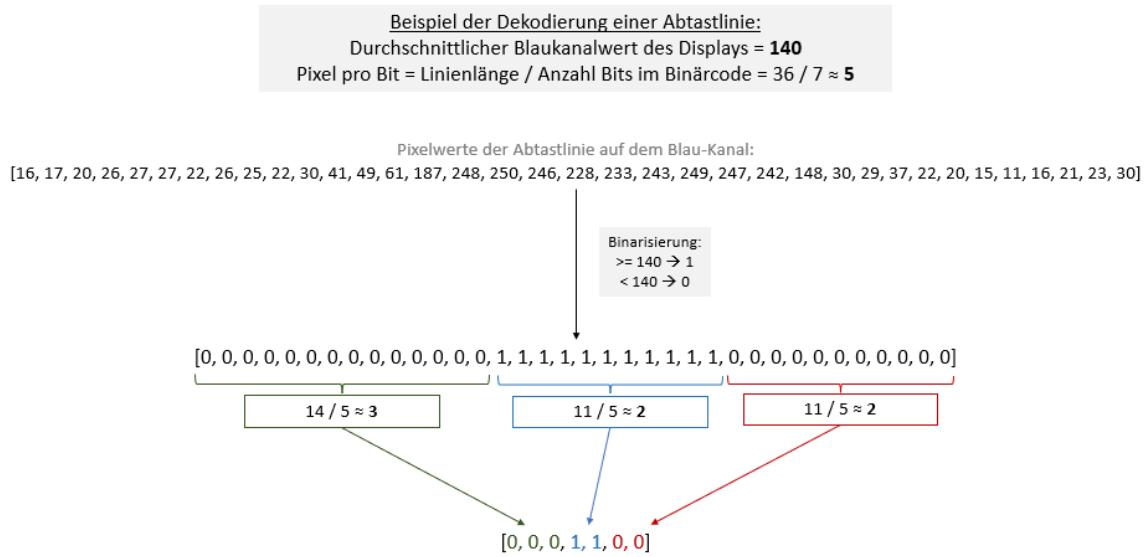


ABBILDUNG 3.14: Beispiel der Dekodierung einer Abtastlinie.

Abbildung 3.14 zeigt die Dekodierung einer Abtastlinie von dem Barcode aus Abbildungen 3.12 und 3.13.

### 3.7.4 Abstimmung aller Abtastlinien

Unter allen Abtastlinien muss abgestimmt werden, welcher Binärkode der Richtige ist. Außerdem ist ein Zuversichtswert für diese Aussage von Vorteil. Aufgrund der Heterogenität der beiden entwickelten Barcode-Designs, muss die Abstimmung an das jeweilige Barcode-Design angepasst werden.

#### Abstimmung aller Abtastlinien: Barcode-Design 1

Ein wichtiger Aspekt des ersten Barcode-Designs ist, dass die Abtastlinien im unteren blauen Bildschirmbereich die Binärschreibweise der Roboternummer invers darstellen. Diese müssen entsprechend invertiert werden. Zu Beginn wird bei einer ungeraden Anzahl an Abtastlinien die mittlere dupliziert. Anschließend wird für die obere und die untere Hälfte geprüft, welches das erste Bit ihrer Binärcodes ist, um über eine Abstimmung festzustellen, welches Startbit für die obere Hälfte steht. Alle Binärcodes, die ein anderes Startbit aufweisen, und damit in der unteren Hälfte des Bildschirms liegen, werden invertiert.

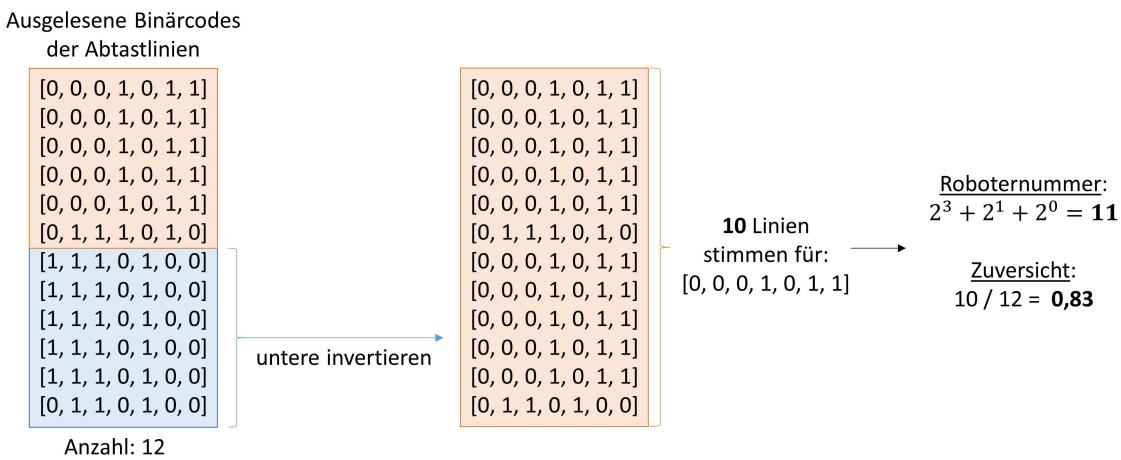


ABBILDUNG 3.15: Beispiel der Abstimmung mit Barcode-Design 1.

Im Anschluss kann eine Abstimmung aller Binärcodes stattfinden. Der Binärkode, den die meisten Abtastlinien erkannt haben, wird zum finalen Binärkode gewählt.

## KAPITEL 3. METHODENENTWICKLUNG

Der Anteil dieser Abtastlinien an der Gesamtzahl der Abtastlinien bildet die Zuversicht dieses Ergebnisses. Dieser finale Binärkode wird ins Dezimalsystem umgewandelt und ergibt die Roboternummer.

Abbildung 3.15 zeigt ein Beispiel für eine solche Abstimmung. Es handelt sich um die ausgelesenen Bitsequenzen der Abtastlinien von dem Barcode in Abbildung 3.10. Insgesamt wurden für diesen Bildschirm zwölf Abtastlinien erstellt und dekodiert. Die unteren sechs Binärcodes müssen invertiert werden. Die darauf folgende Abstimmung ergibt, dass zehn der zwölf Abtastlinien für die Sequenz „0001011“ stimmen. Durch Umwandlung in Dezimalschreibweise ergibt sich daraus die Roboternummer 11 mit einer Zuversicht von 0,83.

### **Abstimmung aller Abtastlinien: Barcode-Design 2**

Für die Abstimmung bei Nutzung des zweiten Barcode-Designs müssen keine Abtastlinien invertiert werden. Trotzdem werden die oberen Abtastlinien separat von den unteren betrachtet. In beiden Hälften wird jeweils eine Abstimmung durchgeführt. Die Zuversicht entspricht der Anzahl Abtastlinien, die für den Gewinner-Code ihrer Hälfte abgestimmt haben, geteilt durch die Gesamtanzahl an Abtastlinien. Steht die Reihenfolge der Bits für die obere und die untere Zeile des blauen Bildschirmbereichs fest, kann diese in die Roboternummer übersetzt werden. Alle Bedingungen werden überprüft. Wird eine Bedingung nicht erfüllt, so gilt die gelesene Roboternummer als ungültig.

Abbildung 3.16 stellt ein Beispiel für die Abstimmung mit Barcode-Design 2 zwar. Die Binärcodes stammen von den Abtastlinien, welche für den Barcode aus Abbildung 3.11 erstellt wurden. Die obere und die untere Hälfte der Binärcodes halten jeweils eine separate Abstimmung. Sieben der acht oberen Binärcodes stimmen für „0101011“. Dieser präsentiert das Bitmuster der oberen Zeile des Barcodes. Ebenfalls sieben der acht unteren stimmen für „1001100“, welcher die untere Zeile des Barcodes darstellt. Mit dem Wissen, wie der Barcode aussieht und wie sein Design erarbeitet wurde, kann man die Roboternummer feststellen.

Wie in Abschnitt 3.6.2 beschrieben, wird in der unteren Zeile des Barcodes geprüft, ob der rechte Block der Blockpaare leuchtet und in der oberen der Linke. Für das

### 3.8. ÜBERARBEITUNG DER ERGEBNISSE

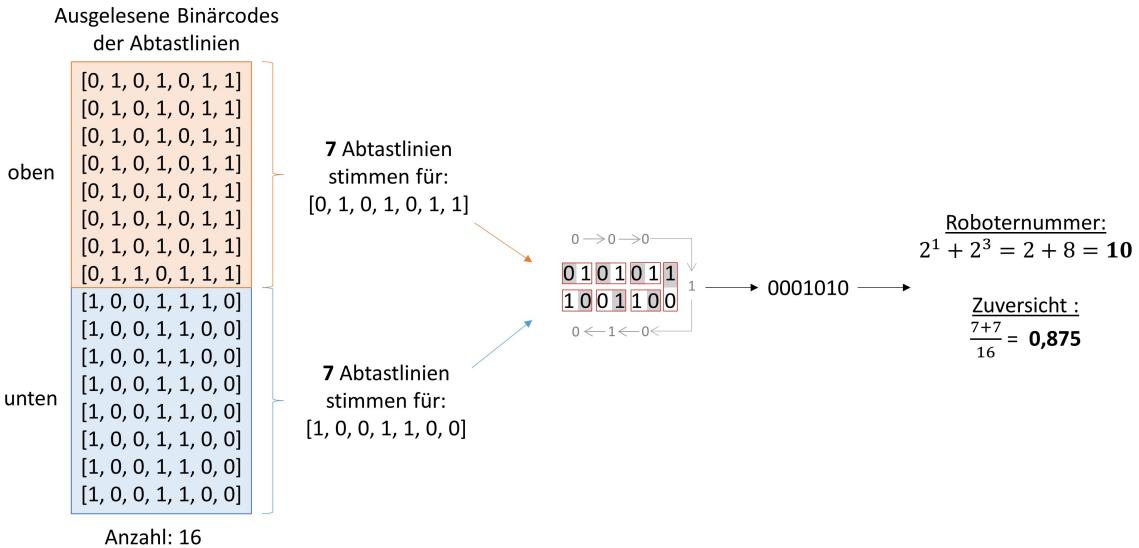


ABBILDUNG 3.16: Beispiel der Abstimmung mit Barcode-Design 2.

Blockpaar rechts außen zählt der obere Block. Die zusammen betrachteten Blöcke sind im Beispiel rot umrandet. Die Blöcke, deren Bits übernommen werden, wurden grau hinterlegt. Um die Barcode-Blöcke herum, stehen in grau diese resultierenden Bits, die Reihenfolge ist durch die Pfeile angegeben. Es ergibt sich der Binärcode „0001010“, welcher durch Umrechnung in Dezimalschreibweise die Roboternummer 10 ergibt. Da sieben der oberen und sieben der unteren Binärcodes der insgesamt 16 für dieses Ergebnis verantwortlich waren, liegt die Zuversicht bei 0,875.

## 3.8 Überarbeitung der Ergebnisse

Zur Steigerung der Effizienz und Präzision werden die Ergebnisse automatisch nachbearbeitet und die Funktionsaufrufe optimiert. Die Nachbearbeitung beinhaltet das Verfolgen der einzelnen Dezibots durch Festlegen eines möglichen Bewegungsbereichs. Dabei werden die Roboter auf dem neuen Bild gesucht und mit der zuvor ermittelten Roboterliste verglichen. Liegen die Standorte sehr nah beieinander, handelt es sich um den gleichen Roboter. Die maximale Distanz ist auf 55 Längeneinheiten bei euklidischer Distanzberechnung gesetzt. Dies entspricht in etwa dem Radius eines Roboters auf dem Bild, wodurch es nicht zu Verwechslungen kommen kann.

## KAPITEL 3. METHODENENTWICKLUNG

Bei Robotern, die einmal mit hoher Zuversicht identifiziert wurden, muss der Barcode auf dem Roboterbildschirm nicht erneut dekodiert werden. Die Dekodierung des Barcodes wie beschrieben in Abschnitt 3.7 erfolgt nur dann, wenn die Zuversicht des vorherigen Barcodes unter 85% liegt. In diesem Fall wird zusätzlich überprüft, ob die neu gefundene Roboternummer eine höhere Zuversicht aufweist, als die vorherige. Wenn nicht, wird die vorherige Roboternummer beibehalten. Die Informationen werden in der Roboterliste aktualisiert. Roboter, die zu keinem der vorher identifizierten Roboter passen, werden neu in die Roboterliste aufgenommen.

### 3.9 Anzeigen und Protokollieren der Ergebnisse

Die Resultate des Trackings können graphisch auf dem entzerrten Arena-Bild festgehalten werden. Die Roboterstandorte entstehen durch Verschiebung des Mittelpunkts des gelben Bereichs um zehn Längeneinheiten in Richtung der Orientierung des Roboters.

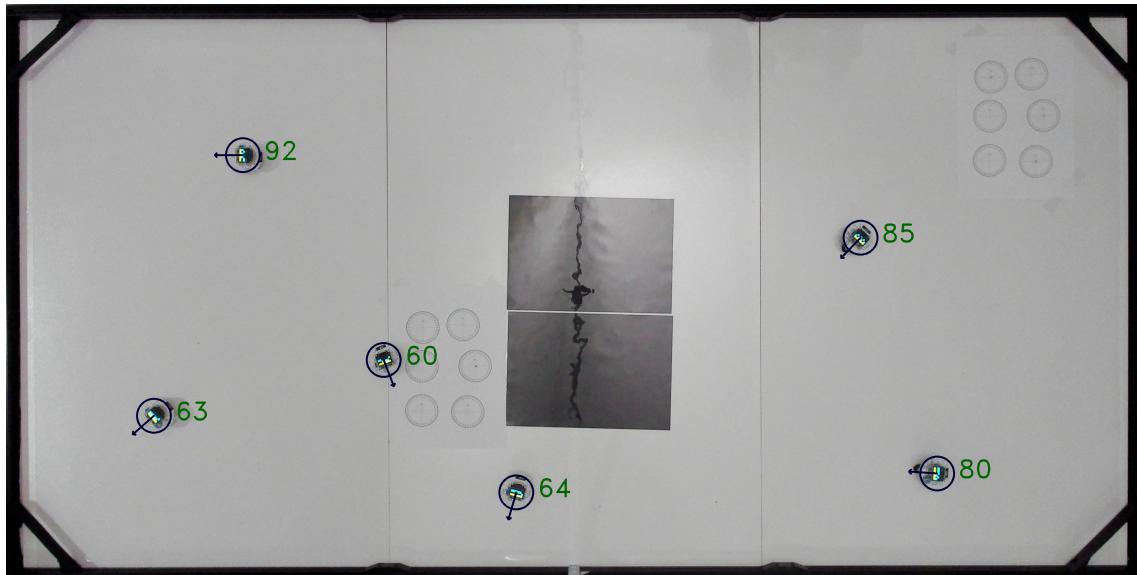


ABBILDUNG 3.17: Arena-Aufnahme mit eingezeichneten Informationen über die Roboter.

Um jeden gefundenen Robotermittelpunkt wird ein Kreis im Radius von 60 Längeneinheiten eingezeichnet. Die Orientierung des Roboters wird durch einen Pfeil

### 3.10. LAUFZEITANALYSE MITHILFE VON PROFILERN

dargestellt, beginnend vom Robotermittelpunkt. Die Roboternummer wird neben den Roboter geschrieben. Die Farbe der Roboternummer gibt an, wie hoch die Zuversicht dieser Roboternummer ist. Bei einer Zuversicht über 60% wird die Farbe grün verwendet, zwischen 40% und 60% orange. In dem Fall, dass die Zuversicht geringer ausfällt, wird die Roboternummer nicht angezeigt. Abbildung 3.17 zeigt, wie ein Bild der Arena mit den eingezeichneten Ergebnissen aussieht.

Eine weitere Option ist das Protokollieren der erkannten Informationen. Für jedes Bild wird die genaue Uhrzeit des Eintrags mit Positionen, Orientierungen, Roboternummern und deren Zuversichten von allen Robotern vermerkt. Das Protokoll kann zur Auswertung von Experimenten genutzt werden.

## 3.10 Laufzeitanalyse mithilfe von Profilern

Zur detaillierten Analyse der Laufzeit und der Aufrufe einzelner Funktionen wurde „cProfiler“ genutzt. Dieser durchläuft das angegebene Programm und speichert eine Statistik mit Informationen zur Laufzeit. Anschließend wurde „gprof2dot“ verwendet, um die Statistik des Profilers in einem Graph aufzubereiten. Dieser Graph wird in einem Dot-Dokument gespeichert, welches man sich mit dem Programm XDOT anzeigen lassen kann.

Abbildung 3.18 zeigt den Graph, der bei der Analyse des Tracking-Systems entsteht. Analysiert wurden die ersten 500 Bilder eines Videos mit sechs Robotern, die sich zufällig über die Arena bewegen. Nicht alle Funktionen erscheinen im Graph, nur die Rechenaufwendigsten. In der ersten Zeile einer jeden Box steht der Name der Funktion. In der zweiten Zeile steht, welchen Anteil der Gesamtrechenleistung die Funktion beansprucht inklusive weiterer Funktionen, die diese Funktion aufruft. Darunter in Klammern ist vermerkt, wie viel Prozent der Gesamtrechenleistung die Funktion selbst nutzt, also ausgenommen anderer aufgerufener Funktionen. Unten in jeder Box steht, wie oft die Funktion aufgerufen wurde. Die Farben zeigen an, wie rechenintensiv die jeweilige Funktion ist, von sehr aufwendig in rot bis wenig aufwendig in blau.

Unter Anderem sieht man in der Abbildung, dass das Lesen der Bilder aus der Videoaufnahme (englisch: video capture) etwa 20% der Laufzeit in Anspruch nimmt.

## KAPITEL 3. METHODENENTWICKLUNG

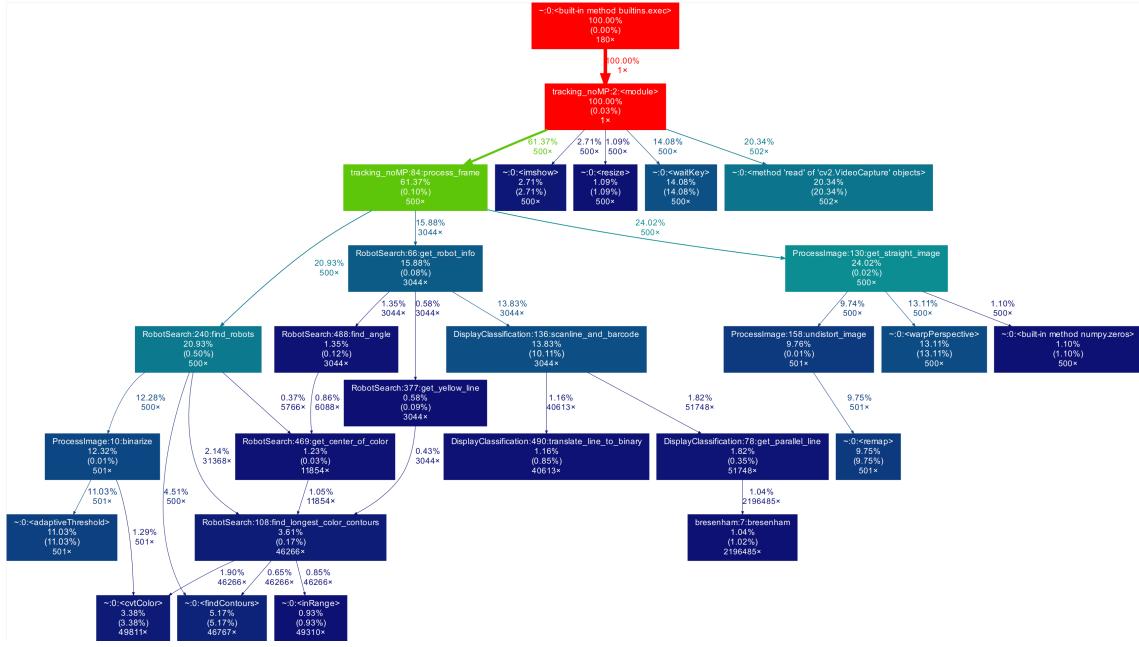


ABBILDUNG 3.18: Laufzeit-Statistik aufbereitet mit DotViewer.

Beim Verarbeiten der einzelnen Bilder durch die Funktion „process\\_frame“ wird circa ein Viertel der Gesamlaufzeit für die Vorverarbeitung mit „get\\_straight\\_image“ aufgebracht. Rund 20% der Laufzeit werden damit verbracht, die Positionen der Roboter auf der Arena zu bestimmen. Die Bestimmung der Orientierungen der Roboter braucht nur wenig Rechenzeit. Die Bestimmung der Roboternummer nutzt insgesamt etwa 14% der Gesamlaufzeit. Interessant sind auch Funktionen wie der Bresenham-Algorithmus oder das Auffinden der Konturen, die besonders häufig aufgerufen wurden.

Während der Entwicklung des Tracking-Systems wurde mehrmals ein solcher Graph erstellt. So konnte festgestellt werden, welche Funktionen besonders aufwendig waren oder sehr häufig aufgerufen wurden. Durch Änderung der Parameterwahl beispielsweise bei dem Binarisieren mit einem adaptiven Schwellenwert kann Laufzeit gespart werden. Häufige Aufrufe können gegebenenfalls reduziert werden, indem einmal berechnete Informationen an folgende Funktionen weitergegeben werden, anstatt sie neu zu berechnen. Für sehr rechenaufwendige Funktionen kann überprüft werden, ob diese vereinfacht oder durch andere Funktionen ersetzt werden können.

## 3.11 Umsetzung der Parallelverarbeitung

Mit dem Ziel, das Programm so effizient wie möglich auszuführen, sollen alle verfügbaren Prozessoren des Rechners in den Verarbeitungsprozess eingebunden werden. Sowohl die Aufnahme eines Bildes in das Programm sowie das Anzeigen des fertigen Bildes auf dem Bildschirm nehmen viel Zeit in Anspruch. Aus diesem Grund wurde dem Einlesen der einzelnen Bilder ein eigener Prozess zugewiesen. Das Anzeigen der fertigen Bilder benötigt geringeren Aufwand und findet auf einem separaten Thread des Hauptprozesses statt.

Außerdem findet die Aktualisierung der Roboterliste auf einem eigenen Prozessor statt, da diese immer auf dem neusten Stand gehalten werden muss. Alle restlichen CPUs des Rechners werden zum Berechnen der Informationen auf den eingehenden Bildern genutzt. Diese Hauptaufgabe nimmt am meisten Rechenzeit in Anspruch und kann durch die Unabhängigkeit der einzelnen Bilder sehr gut parallelisiert werden.

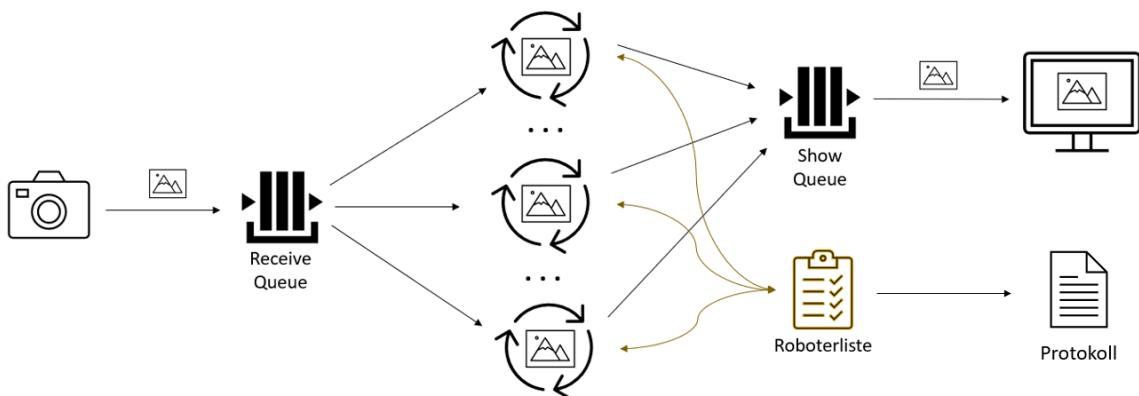


ABBILDUNG 3.19: Arbeitsablauf des Tracking-Systems.

Abbildung 3.19 zeigt den Arbeitsablauf des Programms. Die Kamera nimmt durchgehend auf, alle drei Millisekunden wird ein Bild in eine Warteschlange (englisch: Queue) mit dem Namen „Receive Queue“ gesetzt. Nacheinander werden Bilder aus der Warteschlange von den Prozessen entgegengenommen, die sich mit der Verarbeitung beschäftigen. Nach der Berechnung aller Informationen wird in Rücksprache mit dem Prozess für die Aktualisierung der Roboterliste diese auf den neusten Stand gebracht. Die Roboterliste wird nach jeder Aktualisierung mit einem Zeitstempel im

## *KAPITEL 3. METHODENENTWICKLUNG*

Protokoll vermerkt.

Schließlich wird das fertige Bild, welches alle Informationen graphisch enthält, einer weiteren Warteschlange namens „Show Queue“ angefügt. Der Thread zum Anzeigen der Bilder auf dem Computerbildschirm nimmt nacheinander die Bilder aus der Warteschlange und lässt sie anzeigen. Das Programm wird beendet, wenn die „q“-Taste gedrückt wird oder wenn das zu verarbeitende Video sein Ende erreicht hat.

# 4 Resultate

In diesem Kapitel werden die Ergebnisse von verschiedenen Messungen und Experimenten präsentiert, welche die Güte des Tracking-Systems einordnen. In der Tiefe werden die Genauigkeiten der bestimmten Positionen und Orientierungen, und die Korrektheit der Roboternummern überprüft. Diese werden durch Analysen der aufgezeichneten Protokolle erfasst. Von Interesse ist außerdem die Laufzeit des Systems für verschiedene Anwendungsszenarien.

## 4.1 Präzision der Positionsbestimmung

Die erste zu erfüllende Aufgabe des Tracking-Systems ist die Lokalisierung aller Roboter in der Arena. Dabei spielen vor Allem zwei Kriterien eine Rolle: die Genauigkeit und die Beständigkeit der Erkennung. Erstere wurde durch Messungen in der physischen Arena überprüft. Letztere über die Analyse von Protokollen eines aufgezeichneten Videos.

### 4.1.1 Erkennungsfehler bei der Positionsbestimmung

Um den Erkennungsfehler bei der Bestimmung der Positionen einzuordnen, wurden die Positionen von zwanzig Robotern in der Arena händisch gemessen. Es wurde darauf geachtet, die Messungen in unterschiedlichen Gebieten der Arena vorzunehmen. Für jede Messung wurde ein Video aufgenommen, woraus die mittlere Position ermittelt wird. Der Erkennungsfehler kann über die euklidische Distanz zwischen der gemessenen Position und der vom System ermittelten Position festgestellt werden.

## KAPITEL 4. RESULTATE

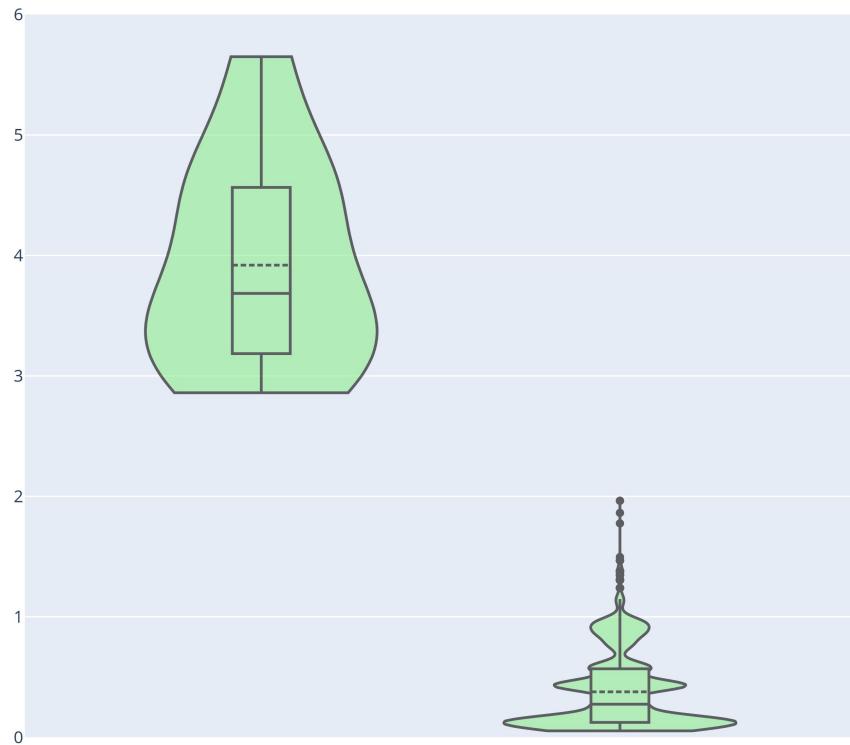


ABBILDUNG 4.1: Verteilung der Erkennungsfehler in Zentimetern bei der Positionsbestimmung mit 20 Messungen (links) und Verteilung der Abweichungen der erkannten Positionen zur mittleren Position in Zentimetern mit fünf unbewegten Robotern auf 1.357 Bildern (rechts).

Die Verteilung der Erkennungsfehler stellt Abbildung 4.1 auf der linken Seite in einem Violinendiagramm dar. Der Median dieser Erkennungsfehler liegt bei 3,69 Zentimetern (cm). Minimum und Maximum liegen bei 2,86 cm und 5,65 cm. Der Abstand zwischen Minimum und Maximum beträgt also weniger als drei Zentimeter. Die mittleren 50% der sortierten Erkennungsfehler liegen im so genannten Interquartilsabstand, also zwischen dem ersten Quartil bei 25% und dem dritten Quartil bei 75%. Dieser wird im Violinendiagramm als Rechteck um den Median dargestellt. Die gestrichelte Linie steht für den Mittelwert der Verteilung. Anhand der eingenommenen Violinenfläche ist erkennbar, wie hoch die Dichte der Datenpunkte an jeder Stelle ist. Die Violine wird nach unten hin breiter, was bedeutet, dass bei mehr Messungen ein geringerer Erkennungsfehler erfasst wurde.

## 4.2. PRÄZISION DER BERECHNUNG DER ORIENTIERUNG

### 4.1.2 Beständigkeit der Positionsbestimmung

Um die Beständigkeit der bestimmten Positionen auf den Prüfstand zu stellen, wird das Protokoll eines Videos mit 1.374 Bildern verwendet. In diesem Video werden die Roboter nicht bewegt, die Positionen bleiben unverändert.

Anhand einer solchen Aufnahmen kann analysiert werden, wie stark die ermittelten Positionen variieren. Dafür wird die mittlere Position eines Roboters über alle Bilder berechnet und für jedes Bild die Abweichung zu dieser mittleren Position über die euklidische Distanz festgestellt. Es werden fünf Roboter betrachtet, insgesamt ergeben sich somit 6.785 Positionen.

Abbildung 4.1 zeigt auf der rechten Seite die Verteilung dieser Abweichungen zur mittleren Position. Der Median der Verteilung liegt bei 0,63 cm, das Minimum bei 0,19 cm. Einige Ausreißer sind oberhalb der Violine als Punkte dargestellt. Doch selbst die maximale Abweichung mit 1,96 cm übersteigt nicht den Radius der Roboteroberfläche, der bei etwa 2,75 Zentimetern liegt. Somit ist sichergestellt, dass es nicht zu Verwechslungen kommen kann.

## 4.2 Präzision der Berechnung der Orientierung

Nach den Positionen werden die Orientierungen aller Roboter berechnet. Auch hier sind die Erkennungsfehler und die Beständigkeit der Berechnung von Interesse. Zur Einordnung wurde das gleiche Vorgehen wie für die Positionsbestimmung genutzt. Für beide Experimente, die in Abschnitt 4.1 beschrieben sind, wurden ebenfalls Messungen für die Orientierungen der Roboter vorgenommen.

### 4.2.1 Erkennungsfehler bei der Berechnung der Orientierung

Zur Messung der Erkennungsfehler bei der Berechnung der Orientierung wurden 20 Roboter auf Winkelkreise gestellt. So kann der Orientierungswinkel des Roboters direkt abgelesen werden. Das vordere Bein des Roboters zeigt genau in Laufrichtung, welche senkrecht zum gelben Balken des Roboterbildschirms gerichtet ist.

## KAPITEL 4. RESULTATE

Von diesen aufgestellten Robotern wurden Videos aufgezeichnet, in denen sie sich nicht bewegen. Für alle Bilder der Videos wird vom Tracking-System eine Orientierung für jeden Roboter berechnet. Aus diesen ermittelten Orientierungen ergibt sich die mittlere Orientierung eines jeden Roboters.

Von Interesse ist die Differenz zwischen dem real gemessenen Orientierungswinkel und dem vom System ermittelten Orientierungswinkel. Die Verteilung dieser Differenzen wird in Abbildung 4.2 auf der linken Seite dargestellt.

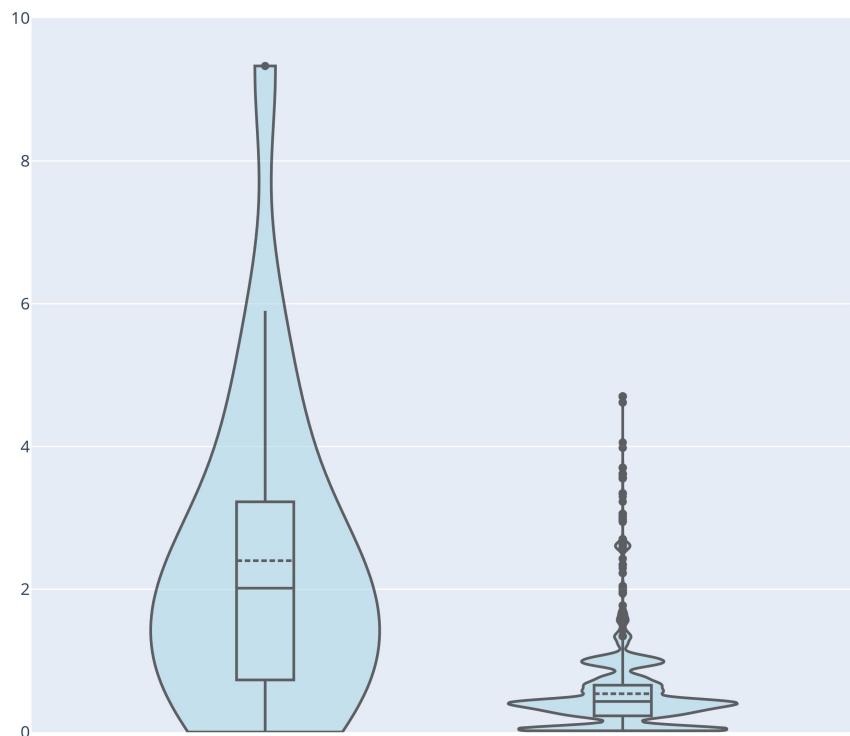


ABBILDUNG 4.2: Verteilung der Erkennungsfehler bei der Berechnung der Orientierungen in Grad mit 20 Messungen (links) und Verteilung der Differenzen der erkannten Winkel zum mittleren Winkel in Grad mit fünf unbewegten Robotern auf 1.357 Bildern (rechts).

Der Median der Verteilung liegt bei 2,0 Grad, der Mittelwert bei 2,4 Grad. Der minimale Fehler liegt bei null, der maximale Fehler ist ein Ausreißer bei 9,3 Grad. Dieser liegt weit entfernt von allen anderen Datenpunkten.

### *4.3. PRÄZISION UND ZUVERSICHTEN DER BARCODE-DEKODIERUNG*

#### **4.2.2 Beständigkeit bei der Berechnung der Orientierung**

Auch bei der Berechnung der Orientierungen ist eine robuste Messung wünschenswert. Wie in Abschnitt 4.1.2 wurde für die Analyse der Beständigkeit das Protokoll eines Videos mit 1.357 Bildern ausgewertet, in dem sich fünf Roboter nicht bewegen. Für jeden Roboter in dem Video wurde ein mittlerer Orientierungswinkel über alle berechneten Orientierungswinkel gebildet. Von Interesse sind im Folgenden die Differenzen zwischen allen gefundenen Orientierungswinkeln und diesem mittleren Orientierungswinkel für jeden Roboter.

Die Verteilung von allen 6.785 Differenzen wird in Abbildung 4.2 auf der rechten Seite dargestellt. Der Median liegt bei 0,62 Grad, der Durchschnitt bei 0,81 Grad. Die untere Grenze aller Differenzen ist bei 0,07 Grad zu finden, die obere Grenze bei 4,3 Grad. Die Mehrheit der Datenpunkte weist eine Differenz unterhalb von zwei Grad auf.

### **4.3 Präzision und Zuversichten der Barcode-Dekodierung**

Schließlich werden die Barcodes der Roboterbildschirme dekodiert. Anders als bei der Positions- und Orientierungsbestimmung lässt sich hier keine Genauigkeit berechnen. Es lässt sich nur feststellen, ob die erkannte Roboternummer richtig oder falsch ist und mit welcher Zuversicht das Tracking-System diese Entscheidung getroffen hat. Dementsprechend sind auch die durchgeführten Versuche für diesen Teil des Trackings neu.

Zuerst wird der prozentuale Fehler der Roboternummererkennung festgestellt. Dabei wird darauf eingegangen, welchen Unterschied die Nutzung der Ergebnisüberarbeitung und der verschiedenen Barcode-Designs macht.

Anschließend liegt der Fokus auf den Zuversichten der erkannten Roboternummern. Auch hier wird ein Vergleich zwischen den Ergebnissen mit und ohne Überarbeitung gezogen und die Barcode-Designs werden miteinander verglichen. Schließlich

## KAPITEL 4. RESULTATE

werden die Zuversichten der richtig und falsch erkannten Roboternummern gegenübergestellt.

### 4.3.1 Präzision der Barcode-Dekodierung

Um die Fehlerrate der Barcode-Dekodierung einzuordnen, wurden jeweils zwei Experimente mit beiden Barcode-Designs durchgeführt. Beim Ersten werden die Roboternummern regelmäßig inkrementiert und die Roboter nicht bewegt. Beim Zweiten bleiben die Roboternummern bestehen und die Roboter bewegen sich zufällig über die Arena. Letzteres kommt einem realen Szenario für ein Experiment nah.

#### Präzision bei Inkrementierung der Roboternummer

Um zu überprüfen, ob alle möglichen Roboternummern vom Tracking-System erkannt werden können, wurden die Roboter so programmiert, dass ihre Roboternummer regelmäßig um eins inkrementiert wird. Die Anzeige des Roboterbildschirms ändert sich entsprechend. Dank diesem Vorgehen können alle darstellbaren Roboternummern analysiert werden. Die Roboter bewegen sich nicht, aber wurden in verschiedenen Regionen der Arena positioniert.

Von den Robotern wurden Videos aufgenommen, welche vom Tracking-System analysiert wurden. Die Ergebnisse wurden in Protokollen vermerkt. Das Video, das für Barcode-Design 1 genutzt wird, beinhaltet 2.035 Bilder mit neun Robotern. Somit sind im Protokoll 18.315 Roboternummern vermerkt. Das Video mit Barcode-Design 2 besteht aus 1.971 Bildern mit je acht Robotern. Entsprechend sind 15.768 Roboternummern im Protokoll.

Das händische Überprüfen der Roboternummern würde nicht nur viel Zeit in Anspruch nehmen, sondern auch nur eine geringe Stichprobe umfassen können. Deshalb wurden die Protokolle automatisch analysiert. Die Nummern auf dem ersten Bild werden von einer Person abgelesen und dem Programm übermittelt. Anschließend gilt eine Nummer als korrekt, wenn sie der Vorherigen entspricht oder um eins oder zwei höher als die Vorherige ist.

### 4.3. PRÄZISION UND ZUVERSICHTEN DER BARCODE-DEKODIERUNG

Ein Sonderfall ist für Barcode-Design 1 die Roboternummer 126, die folgende Roboternummer ist hier die 1. Für Barcode-Design 2 spielt die Roboternummer 127 eine besondere Rolle, ihr folgt die 0.

Selbstverständlich ist die Überarbeitung der Ergebnisse für dieses Experiment ausgeschaltet.

	Barcode-Design 1	Barcode-Design 2
richtig erkannt	96,0% (17.587)	91,3% (14.399)
falsch erkannt	4,0% (728)	8,7% (1.369)
- davon nicht gelesen	0,0% (7)	8,5% (1.339)
- davon falsche Reihenfolge	1,9% (339)	0,2% (28)
- davon ungeklärter Ursache	2,1% (382)	0,0% (2)

TABELLE 4.1: Präzision der Barcode-Dekodierung bei Inkrementierung der Roboternummer unter Verwendung beider Barcode-Designs.

Tabelle 4.1 zeigt die Resultate der Analysen. Mit dem ersten Barcode-Design konnten 96,0% der Roboternummern richtig erkannt werden, mit dem zweiten 91,3%. Die Fehlerrate ist demnach beim zweiten Barcode-Design mehr als doppelt so groß wie beim ersten.

Dafür lassen sich die Fehler beim zweiten Barcode-Design fast vollständig erklären. Von den 8,7% falsch erkannter Roboternummern wurde der Großteil mit 8,5% nicht gelesen, weil nicht alle Bedingungen des Designs erfüllt waren. Lediglich 0,2% sind verschuldet durch eine falsche Reihenfolge, also dass beispielsweise die Roboternummer 11 statt der 12 erkannt wurde. In zwei Fällen kann die Fehlerursache nicht geklärt werden. Für Experimente ist es besser zu wissen, dass ein Barcode nicht dekodiert werden konnte, als falsche Resultate zu erhalten.

Mit Barcode-Design 2 gab es nur sieben Fälle, in denen der Barcode nicht gelesen werden konnte. Das passiert beispielsweise, wenn der gelbe Balken nicht gefunden werden kann. Mit 1,9% tritt hier der Fall auf, dass die Reihenfolge der Roboternummern vertauscht ist. Die Mehrheit der falsch erkannten Roboternummern, nämlich 2,1% lassen sich nicht erklären.

## KAPITEL 4. RESULTATE

### Präzision bei Bewegung der Roboter

Im Fall eines Schwarmintelligenz-Experiments behalten alle Roboter ihre Roboternummer bei und bewegen sich über die Arena. In einer weiteren Analyse wird dieses Szenario geprüft. Dabei soll der Effekt der Überarbeitung der Ergebnisse und der Unterschied der beiden Barcode-Designs erfasst werden.

Für das erste Barcode-Design wurde ein Video mit 1.826 Bildern und sechs Robotern aufgenommen und ausgewertet. Das Video für Barcode-Design 2 enthält 956 Bilder und ebenfalls sechs Roboter. Insgesamt gibt es somit 10.956 Roboternummern im Protokoll für Barcode-Design 1 und 5.736 für Barcode-Design 2.

Für das Experiment werden die Roboternummern der sechs Roboter angegeben und überprüft, auf wie vielen Bildern diese Roboternummern erkannt wurden. Bei beiden Videos handelt es sich um die gleichen Roboter mit den gleichen Roboternummern.

	Barcode-Design 1	Barcode-Design 2
ohne Überarbeitung	98,4%	80,0%
mit Überarbeitung	<b>99,5%</b>	99,4%

TABELLE 4.2: Präzision der Barcode-Dekodierung mit und ohne Überarbeitung der Ergebnisse unter Verwendung beider Barcode-Designs.

Tabelle 4.2 zeigt den prozentualen Anteil aller richtig erkannten Roboternummern für beide Barcode-Designs mit und ohne Überarbeitung der Ergebnisse. Vor Allem Barcode-Design 2 profitiert stark von der Überarbeitung der Ergebnisse. Werden die Ergebnisse wie in Abschnitt 3.8 beschrieben überarbeitet, sind die Präzisionen beider Barcode-Designs nahezu gleich. Ohne die Überarbeitung hat das erste Barcode-Design gegenüber dem zweiten einen deutlichen Vorteil von über 18%.

### 4.3.2 Zuversichten der erkannten Roboternummern

Während jeder Barcode-Dekodierung wird eine Zuversicht berechnet, mit der die Entscheidung getroffen wurde, um welche Roboternummer es sich handelt. Sie beruht auf der prozentualen Anzahl der Abtastlinien, die für die gewählte Roboternummer gestimmt haben. Sie liegt immer zwischen null und eins.

#### 4.3. PRÄZISION UND ZUVERSICHTEN DER BARCODE-DEKODIERUNG

Über die Protokolle, die in Abschnitt 4.3.1 erstellt wurden, werden zusätzlich die ermittelten Zuversichten analysiert. Zum einen wird die Verteilung der Zuversichten von richtig erkannten Roboternummern verglichen mit der von falsch erkannten Roboternummern. Zum Anderen werden die Verteilungen der Zuversichten analysiert, wenn eine Überarbeitung der Ergebnisse stattfindet und wenn nicht. In beiden Experimenten wird zwischen den beiden Barcode-Designs unterschieden.

##### Zuversichten von richtig und falsch erkannten Roboternummern

Für die Untersuchung der Zuversichten von richtig und falsch erkannten Roboternummern wurden die Videos genutzt, in denen die Roboternummern inkrementiert werden. Wurde eine Roboternummer richtig erkannt, wird ihre Zuversicht einer Liste mit Zuversichten der richtig erkannten Roboternummern hinzugefügt. Auch für falsch erkannte Roboternummern wird eine Liste erstellt.

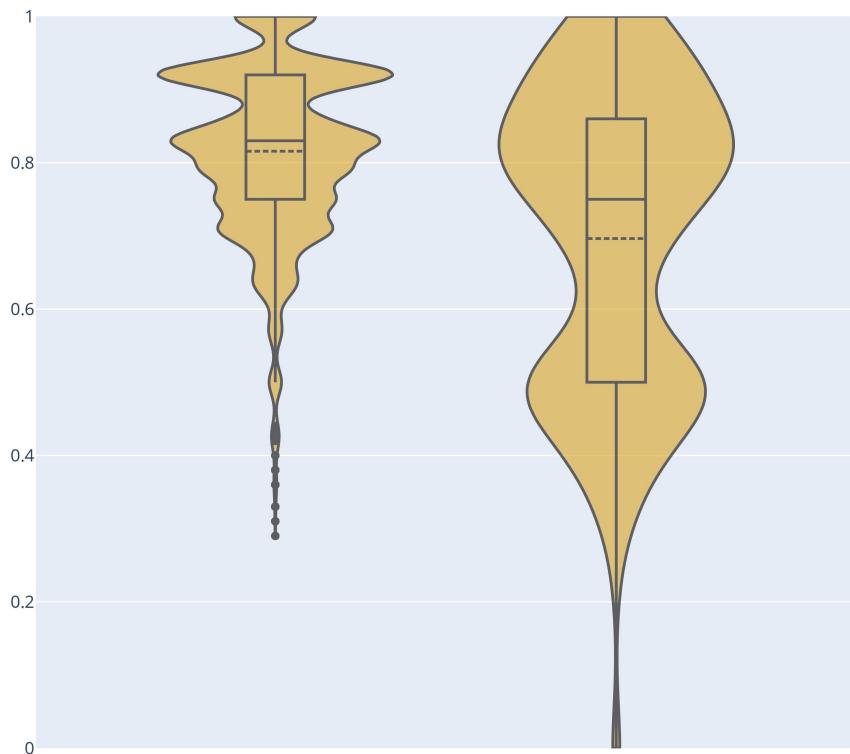


ABBILDUNG 4.3: Zuversichtsverteilung der richtig (links) und der falsch (rechts) erkannten Roboternummern unter Verwendung von Barcode-Design 1.

## KAPITEL 4. RESULTATE

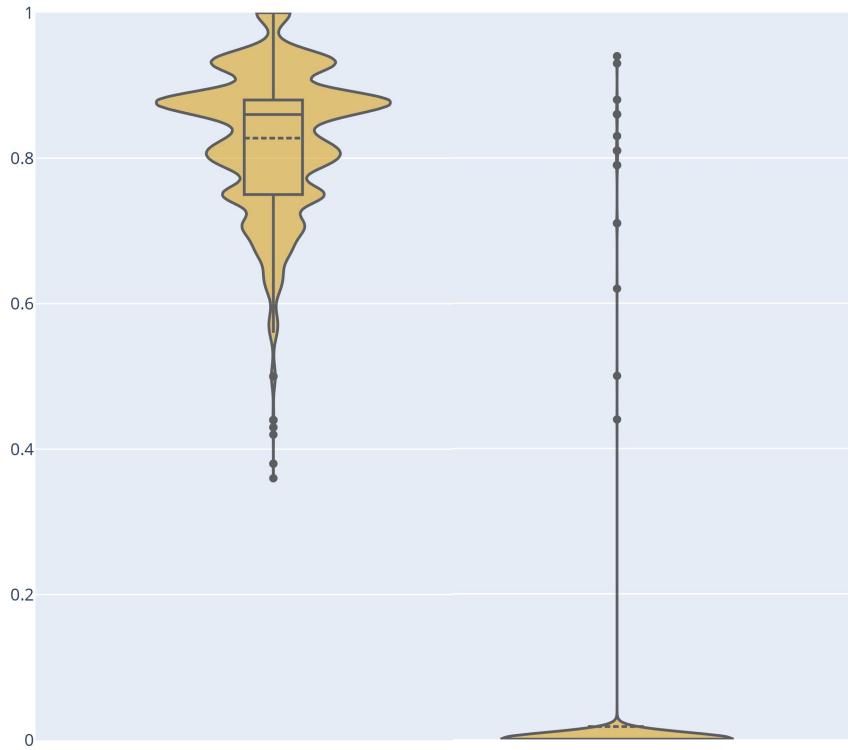


ABBILDUNG 4.4: Zuversichtsverteilung der richtig (links) und der falsch (rechts) erkannten Roboternummern unter Verwendung von Barcode-Design 2.

Die Abbildung 4.3 zeigt die Verteilungen der Zuversichten der richtig und falsch erkannten Roboternummern für das erste Barcode-Design. Analog dazu zeigt Abbildung 4.4 die Verteilungen für das zweite Barcode-Design.

In beiden Abbildungen wird deutlich, dass die Zuversichten für korrekt erkannte Roboternummern im Durchschnitt höher liegen als Zuversichten von falsch erkannten Roboternummern. Bei Verwendung des zweiten Barcode-Designs ist der Unterschied weitaus deutlicher.

Es sollte beachtet werden, dass Barcode-Design 1 in der Analyse nur etwa halb so viele Fehlerkennungen aufweist wie Barcode-Design 2, siehe Tabelle 4.1. Dementsprechend ist die Stichprobe für die Zuversichten von falsch erkannten Roboternummern deutlich kleiner für Barcode-Design 1 als für Barcode-Design 2. Außerdem sind 98% der Fehlerkennungen bei Barcode-Design 2 verschuldet durch nicht ausgelesene Barcodes, wo die Bedingungen des Barcode-Designs nicht erfüllt sind. In diesem Fall, wird die Zuversicht auf null gesetzt. Daher überrascht es nicht, dass der Median der

#### *4.3. PRÄZISION UND ZUVERSICHTEN DER BARCODE-DEKODIERUNG*

Zuversicht der falsch erkannten Roboternummern bei Verwendung von Barcode-Design 2 bei null liegt.

Bei beiden Barcode-Designs liegen Mittelwert und Median der Zuversichten der richtig identifizierten Roboternummern über 0,8. Die Barcode-Designs haben außerdem gemein, dass die Zuversichten der falsch erkannten Roboternummern fast das gesamte Spektrum von null bis eins abbilden. Die Verteilungen machen deutlich, dass der Zuversichts-Wert beim ersten Barcode-Design wenig Ausschluss darüber liefert, ob eine Roboternummer richtig oder falsch ist. Mit Barcode-Design 2 kann man mit hoher Gewissheit feststellen, ob ein Barcode richtig dekodiert wurde oder nicht.

#### **Zuversichten mit und ohne Überarbeitung der Ergebnisse**

Bei der Überarbeitung der Ergebnisse wird für Roboter, bei denen die Roboternummer mit hoher Zuversicht bestimmt wurde, diese Roboternummer nicht neu berechnet sondern inklusive der Zuversicht beibehalten. Wird die Roboternummer neu berechnet, aber war die Zuversicht im vorherigen Bild höher, so wird die vorher berechnete Roboternummer und deren Zuversicht beibehalten.

Wie man in den Abbildungen 4.5 und 4.6 sieht, hat dieses Vorgehen zur Folge, dass die ermittelten Roboternummern nach der Überarbeitung stets sehr hohe Zuversichten haben. Der Median liegt bei beiden Barcode-Designs mit der Überarbeitung bei 1. Nur wenige Ausreißer sind vorhanden.

Die Verteilungen, die entstehen, wenn man die Ergebnisse nicht überarbeitet, unterscheiden sich stark. Betrachtet man die von Barcode-Design 2, so sieht man, dass alle Zuversichten entweder sehr hoch um die 0,9 oder sehr niedrig um 0 sind. In den Fällen, wo der Barcode nicht gelesen werden kann, liegt die Zuversicht bei 0.

Für Barcode-Design 1 liegen die meisten Zuversichten ebenfalls relativ hoch -durchschnittlich bei 0,77- wenn man die Ergebnisse nicht überarbeitet. Allerdings zeichnet sich eine andere Form der Verteilung ab. Trichterförmig nimmt sie nach unten ab.

Trotz der Unterschiede zeigt sich in beiden Betrachtungen, dass die Überarbeitung der Ergebnisse einen großen Vorteil für die Beständigkeit der Ergebnisse birgt.

## KAPITEL 4. RESULTATE

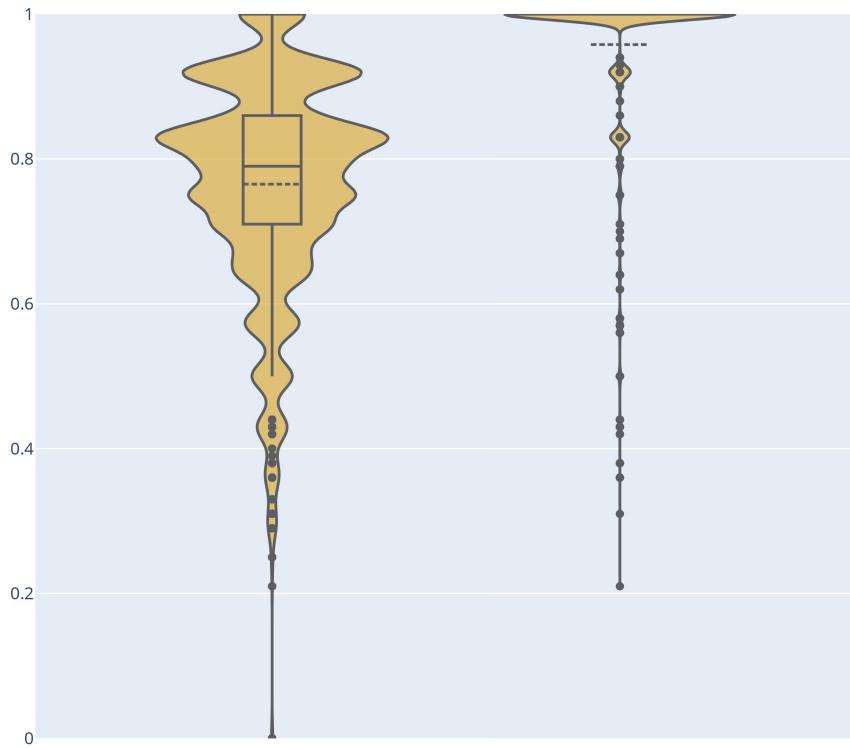


ABBILDUNG 4.5: Zuversichtsverteilung mit (links) und ohne (rechts) Überarbeitung der Ergebnisse unter Verwendung von Barcode-Design 1.

## 4.4 Performanz des Tracking-Systems

Bei Echtzeitanwendungen ist Performanz von hoher Bedeutsamkeit, im besten Fall erscheint das verarbeitete Bild fast zeitgleich auf dem Bildschirm. Bei Bewegungen sollten so viele Bilder pro Sekunde wie möglich verarbeitet und angezeigt werden, damit ein flüssiges Video entsteht.

Im gegebenen Anwendungsfall bewegen sich die Roboter mit geringer Geschwindigkeit, so gibt es wenige Änderungen bei aufeinander folgenden Bildern. Trotzdem sind kurze Laufzeiten für spätere Auswertungen von Vorteil.

Tabelle 4.3 zeigt, wie viele Bilder pro Sekunde (englisch: frames per second, fps) in verschiedenen Szenarien verarbeitet werden können. Die drei verschiedenen Anwendungen sind in den Spalten vermerkt. Man kann sich die Resultate entweder anzeigen lassen, man kann sie protokollieren oder man kann sie sich sowohl auf dem Bildschirm anzeigen als auch in einem Protokoll festhalten.

#### 4.4. PERFORMANZ DES TRACKING-SYSTEMS

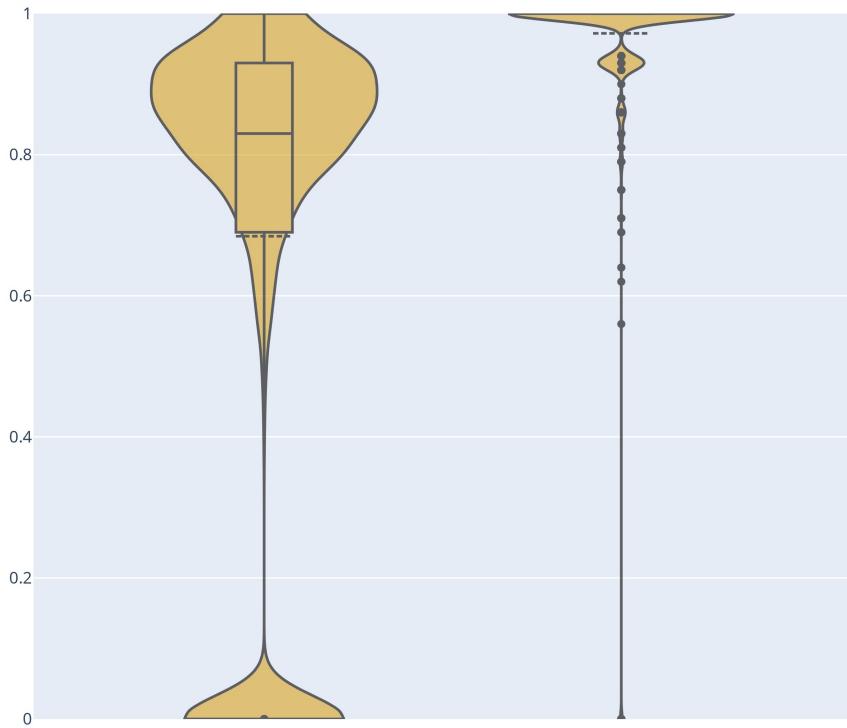


ABBILDUNG 4.6: Zuversichtsverteilung mit (links) und ohne (rechts) Überarbeitung der Ergebnisse unter Verwendung von Barcode-Design 2.

Die vier Zeilen geben an, wie viele logische Prozessoren verwendet wurden und ob die Ergebnisse, wie in Kapitel 3.8 beschrieben, überarbeitet wurden. Bei der Überarbeitung muss der Barcode nicht in jedem Bild neu dekodiert werden, was der Laufzeit zugute kommen sollte. Andererseits sind dafür zusätzliche Vergleiche notwendig.

Die Messungen wurden auf einem Video mit rund 2.000 Bildern durchgeführt. Dafür wurde das Linux-Betriebssystem Ubuntu genutzt. Unter Windows 11 läuft das Tracking-System ebenfalls, allerdings etwas langsamer.

Die Resultate in der Tabelle zeigen, dass die Parallelverarbeitung mit acht logischen Prozessoren einen deutlichen Vorteil birgt. In allen Anwendungsfällen konnten fast doppelt so viele Bilder pro Sekunde verarbeitet werden. Es geht auch hervor, dass beim Protokollieren weniger Zeit verloren geht, aber dass es keinen Unterschied gibt, ob man sich die Resultate nur anzeigen lässt oder zusätzlich noch protokolliert. Auch die Entscheidung, ob die Ergebnisse überarbeitet werden sollen oder nicht, ändert das Ergebnis nicht signifikant. Die meisten Bilder pro Sekunde -nämlich 12,1-, kön-

## KAPITEL 4. RESULTATE

	anzeigen	protokollieren	anzeigen&protokollieren
1 CPU, ohne Überarbeitung	5,5	5,6	5,5
1 CPU, mit Überarbeitung	5,5	5,8	5,4
8 CPUs, ohne Überarbeitung	10,4	11,5	10,4
8 CPUs, mit Überarbeitung	10,5	<b>12,1</b>	10,5

TABELLE 4.3: Vergleich der Anzahl der verarbeiteten Bilder pro Sekunde bei verschiedenen Fällen der Anwendung und Prozessorennutzung.

nen mit acht logischen Prozessoren erreicht werden, wenn die Ergebnisse nur protokolliert werden. Diese Anzahl ist für unseren Anwendungsfall absolut ausreichend.

# 5 Diskussion

Im folgenden Kapitel werden Limitationen der Ergebnisse diskutiert.

Zuerst sollten die Resultate, die sich auf manuelle Abmessungen der Arena stützen, kritisch betrachtet werden, da es schwer ist, die exakte Entfernung zu den Banden auszumessen. Dafür müsste das Maßband in einem perfekten rechten Winkel zur Bande gehalten werden, was nicht garantiert werden kann. Außerdem nimmt bei der Lokalisierung auch die Verzerrung durch die Kamera einen Einfluss. Bei Robotern, die direkt unter der Kamera stehen, ist die Position genauer, als für Roboter in den Ecken, wo die Verzerrung am Stärksten ist.

Auch die erarbeiteten Methoden selbst finden unter gewissen Bedingungen möglicherweise ihre Limitationen. Ein generelles Problem in der Bildverarbeitung stellen Schatten dar. Obwohl das System in verschiedenen Szenarien getestet wurde, lässt sich nicht ausschließen, dass durch neu entstehende Schatten durch Besuchergruppen die Präzision des Tracking-Systems beeinträchtigt werden könnte.

Im Allgemeinen spielen die Lichtverhältnisse eine wichtige Rolle. Sollten in Zukunft Lampen neu aufgestellt oder entfernt werden, kann es potenziell zu einer fehlerhaften Erkennung kommen. Damit einher geht auch die Lage der Arena. Ein anderer Arena-Standort hat andere Lichtverhältnisse, die beachtet werden müssten.

Schließlich ist es denkbar, dass eines Tages neue Versionen des Roboters produziert werden, deren Displays ein anderes Format oder andere Farben haben. Das System müsste entsprechend getestet und gegebenenfalls angepasst werden.

In Anbetracht der parallelen Verarbeitung des Tracking-Systems auf mehreren Prozessoren, spielt offensichtlich die Anzahl der verfügbaren Prozessoren eine entscheidende Rolle. Sind weniger logische CPUs im Gerät vorhanden, so kann die Laufzeit drastisch zunehmen. Förderlich für die Performanz sind nach eigenen Einschätzungen Computer mit vielen Prozessoren.



## 6 Fazit und Ausblick

In dieser Arbeit wurde die Entwicklung eines Tracking-Systems für die ScaDS.AI-Arena beschrieben, mit dem Dezibots anhand ihrer OLED-Bildschirme verfolgt werden können. Die Software ist in der Lage die Roboter präzise zu lokalisieren, ihre Orientierung in Form eines Winkels zu berechnen, und die auf den Bildschirmen gezeigten Barcodes in eindeutige Roboternummern zu dekodieren.

Zum Erreichen der genannten Resultate ist es vorab notwendig, das Kamerabild zu entzerren, die Arena-Ecken zu lokalisieren und eine Homographie durchzuführen, um das Arena-Bild auf diese Arena-Ecken zu ziehen.

Darauf folgt die Lokalisierung der Roboter über Konturen und definierte Kriterien zur eindeutigen Identifizierung. Eine kritische Aufgabe ist die Bewältigung von Überschneidungen in besonderem Anbetracht von Schatten. Die Orientierung der Roboter kann über die gelben Balken der Roboterbildschirme und trigonometrische Betrachtungen berechnet werden.

Eine der Hauptaufgaben der Arbeit stellte die Entwicklung der Barcode-Designs zur graphischen Darstellung der Roboternummer und deren Dekodierung über Abtastlinien dar. Zur Amplifizierung von Performanz und Güte des Tracking-Systems wurden die erfassten Informationen überarbeitet und die Dekodierung nur dann durchgeführt, wenn der Roboter bisher noch nicht sicher erfasst werden konnte.

Die finalen Informationen können auf dem Arenabild visuell aufgezeigt und so live verfolgt werden. Die Analyse eines aufgezeichneten Videos ist ebenfalls umsetzbar. Darüber hinaus ist es möglich, die gefundenen Informationen in einer Protokolldatei zu speichern.

Durch Anwendung von Parallelverarbeitung konnte das Tracking-System erfolgreich an Performanz gewinnen, indem Teilaufgaben auf verschiedene logische Prozessoren verteilt wurden. Profiling wurde als Werkzeug genutzt, um die Laufzeit detailliert

## KAPITEL 6. FAZIT UND AUSBLICK

zu analysieren.

Das Tracking-System dient sowohl als Werkzeug für zukünftige wissenschaftliche Experimente mit den Dezibots als auch zur Vorführung bei Besuch im ScaDS.AI Living Lab.

Zukünftig ist denkbar, das Tracking-System für andere Roboterarten zu erweitern oder für verschiedene Arenen anzupassen. Auch ein Vergleich mit Tracking-Funktionen, die von Bibliotheken zur Verfügung gestellt werden, kann zu interessanten Einblicken in Hinblick auf Performanz und Präzision verhelfen. Die Anwendung von neuronalen Netzen ist durchaus eine Option.

Aufschlussreich wäre außerdem die Entwicklung weiterer Barcodes für die Roboterbildschirme, um diese mit den bereits Entworfenen zu vergleichen. Es ist denkbar, dass eine optimiertere Version mit Fehlererkennung oder sogar -behebung erstellt werden kann.

Das aktuelle System erfüllt alle gestellten Bedingungen mit hoher Präzision, aber könnte in Zukunft noch erweitert werden. Möglicherweise sind zusätzliche Informationen über die Roboter eines Tages von Interesse, die dem System hinzugefügt werden könnten.

Die Arbeit zeigt, dass Bildverarbeitungstechniken zielführend angewendet werden können, um konkrete Informationen aus einem Bild zu ziehen. Dieser Anwendungsfall profitiert stark von der Parallelisierung einzelner Aufgaben auf verschiedenen Prozessoren.

# Abbildungsverzeichnis

2.1	Beispiel Kalibrierungsmatrix . . . . .	6
2.2	Barcode-Standards . . . . .	8
3.1	Originale Arena-Aufnahme . . . . .	14
3.2	Entzerrte Arena-Aufnahme . . . . .	15
3.3	Arena-Aufnahme mit Informationen zur Erkennung der Arena-Ecken	17
3.4	Arena-Aufnahme nach der Homographie . . . . .	18
3.5	Arena-Aufnahme mit allen Konturen und gefundenen Robotern . . .	19
3.6	Gelbe und Blaue Farbmasken . . . . .	20
3.7	Beispiel für die Überschneidung zweier Rechtecke . . . . .	22
3.8	Dezibot mit Winkelkreis . . . . .	24
3.9	Dezibot mit approximierter Mittellinie des gelben Balkens . . . .	25
3.10	Barcode-Design 1 . . . . .	28
3.11	Barcode-Design 2 . . . . .	29
3.12	Dezibot mit eingezeichneter gelber Mittellinie und Abtastlinien . . .	31
3.13	Blaukanal und binarisierte Blaukanal eines Dezibot-Bildausschnitts .	32
3.14	Beispiel der Dekodierung einer Abtastlinie . . . . .	34
3.15	Beispiel der Abstimmung mit Barcode-Design 1 . . . . .	35
3.16	Beispiel der Abstimmung mit Barcode-Design 2 . . . . .	37
3.17	Arena-Aufnahme mit allen Roboter-Informationen . . . . .	38
3.18	Laufzeit-Statistik aufbereitet mit DotViewer . . . . .	40
3.19	Arbeitsablauf des Tracking-Systems . . . . .	41
4.1	Erkennungsfehler und Beständigkeit der Positionsbestimmung . . . .	44
4.2	Erkennungsfehler und Beständigkeit der Berechnung der Orientierungen	46

## *Abbildungsverzeichnis*

4.3	Zuversichtsverteilung der richtig und der falsch erkannten Roboter- nummern unter Verwendung von Barcode-Design 1 . . . . .	51
4.4	Zuversichtsverteilung der richtig und der falsch erkannten Roboter- nummern unter Verwendung von Barcode-Design 2 . . . . .	52
4.5	Zuversichtsverteilung mit und ohne Überarbeitung der Ergebnisse un- ter Verwendung von Barcode-Design 1 . . . . .	54
4.6	Zuversichtsverteilung mit und ohne Überarbeitung der Ergebnisse un- ter Verwendung von Barcode-Design 2 . . . . .	55

# Tabellenverzeichnis

4.1	Präzision der Barcode-Dekodierung bei Inkrementierung der Roboter- nummer unter Verwendung beider Barcode-Designs. . . . .	49
4.2	Präzision der Barcode-Dekodierung mit und ohne Überarbeitung der Ergebnisse unter Verwendung beider Barcode-Designs. . . . .	50
4.3	Vergleich der Anzahl der verarbeiteten Bilder pro Sekunde bei ver- schiedenen Fällen der Anwendung und Prozessorennutzung. . . . .	56



# Literatur

- [1] A. Abraham, H. Guo und H. Liu. „Swarm intelligence: foundations, perspectives and applications“. In: *Swarm intelligent systems*. Springer, 2006, S. 3–25.
- [2] E. Alreshidi, R. A. Ramadan, M. H. Sharif, O. F. Ince und I. F. Ince. „A comparative study of image descriptors in recognizing human faces supported by distributed platforms“. In: *Electronics* 10.8 (2021), S. 915.
- [3] J.-L. Baer. „A survey of some theoretical aspects of multiprocessing“. In: *ACM Computing Surveys (CSUR)* 5.1 (1973), S. 31–80.
- [4] G. Beni. „From swarm intelligence to swarm robotics“. In: *International Workshop on Swarm Robotics*. Springer. 2004, S. 1–9.
- [5] E. Bonabeau, M. Dorigo, G. Theraulaz und G. Theraulaz. *Swarm intelligence: from natural to artificial systems*. Oxford university press, 1999.
- [6] A. Chakraborty und A. K. Kar. „Swarm intelligence: A review of algorithms“. In: *Nature-inspired computing and optimization* (2017), S. 475–494.
- [7] G. Chandan, A. Jain, H. Jain u. a. „Real time object detection and tracking using Deep Learning and OpenCV“. In: *2018 International Conference on inventive research in computing applications (ICIRCA)*. IEEE. 2018, S. 1305–1308.
- [8] C. Chen, W. Li, L. Gao, H. Li und J. Plaza. „Special issue on advances in real-time image processing for remote sensing“. In: *Journal of Real-Time Image Processing* 15.3 (2018), S. 435–438.

## Literatur

- [9] I. Culjak, D. Abram, T. Pribanic, H. Dzapo und M. Cifrek. „A brief introduction to OpenCV“. In: *2012 proceedings of the 35th international convention MIPRO*. IEEE. 2012, S. 1725–1730.
- [10] D. DeTone, T. Malisiewicz und A. Rabinovich. „Deep image homography estimation“. In: *arXiv preprint arXiv:1606.03798* (2016).
- [11] M. Dorigo, G. Theraulaz und V. Trianni. „Swarm robotics: past, present, and future [point of view]“. In: *Proceedings of the IEEE* 109.7 (2021), S. 1152–1165.
- [12] S. U. Dübener, A. Jacker, A. A. Morgner, H. F. Haupt und J. Wagner. „Dezibot<sup>^</sup> sub 2<sup>^</sup>-The Wi-Fi based, inexpensive, and small educational mobile robot“. In: *Proceedings of the International Conference on Frontiers in Education: Computer Science and Computer Engineering (FECS)*. 2018, S. 146–150.
- [13] S. U. Dübener, A. A. Morgner, H. F. Haupt, M. H. Volk, C. C. Fischer, S. K. Langner und A. Jacker. „Gegenüberstellung von kostengünstigen Robotern als Lernobjekte für Schulen“. In: *INFORMATIK 2017* (2017).
- [14] A. Duda und U. Frese. „Accurate Detection and Localization of Checkerboard Corners for Calibration.“ In: *BMVC*. 2018, S. 126.
- [15] R. Girshick, J. Donahue, T. Darrell und J. Malik. „Region-based convolutional networks for accurate object detection and segmentation“. In: *IEEE transactions on pattern analysis and machine intelligence* 38.1 (2015), S. 142–158.
- [16] R. Girshick, J. Donahue, T. Darrell und J. Malik. „Rich feature hierarchies for accurate object detection and semantic segmentation“. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, S. 580–587.
- [17] R. L. Graham. „Bounds on multiprocessing timing anomalies“. In: *SIAM journal on Applied Mathematics* 17.2 (1969), S. 416–429.

- [18] N. Holagundi, G. H. Ashwathsetty und M. Basthikodi. „Algorithm fuzzy scheduling for realtime jobs on multiprocessor systems“. In: *Indonesian Journal of Electrical Engineering and Computer Science* 25.3 (2022), S. 1308–1319.
- [19] A. Ioannou und E. Makridou. „Exploring the potentials of educational robotics in the development of computational thinking: A summary of current research and practical proposal for future work“. In: *Education and Information Technologies* 23.6 (2018), S. 2531–2544.
- [20] P. Janku, K. Koplik, T. Dulik und I. Szabo. „Comparison of tracking algorithms implemented in OpenCV“. In: *MATEC Web of Conferences*. Bd. 76. EDP Sciences. 2016, S. 04031.
- [21] J. Kennedy. „Swarm intelligence“. In: *Handbook of nature-inspired and innovative computing*. Springer, 2006, S. 187–219.
- [22] A. Krizhevsky, I. Sutskever und G. E. Hinton. „Imagenet classification with deep convolutional neural networks“. In: *Communications of the ACM* 60.6 (2017), S. 84–90.
- [23] H. P. Langtangen. *Python scripting for computational science*. Springer, 2008.
- [24] D. Morgan. „Trail pheromones of ants“. In: *Physiological entomology* 34.1 (2009), S. 1–17.
- [25] J. Palach. *Parallel programming with Python*. Packt Publishing Ltd, 2014.
- [26] S. Placht, P. Fürsattel, E. A. Mengue, H. Hofmann, C. Schaller, M. Balda und E. Angelopoulou. „Rochade: Robust checkerboard advanced detection for camera calibration“. In: *European conference on computer vision*. Springer. 2014, S. 766–779.
- [27] F. Remondino und C. Fraser. „Digital camera calibration methods: considerations and comparisons“. In: *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* 36.5 (2006), S. 266–272.

## Literatur

- [28] H. El-Rewini, H. H. Ali und T. Lewis. „Task scheduling in multiprocessing systems“. In: *Computer* 28.12 (1995), S. 27–37.
- [29] M. Schranz, M. Umlauft, M. Sende und W. Elmenreich. „Swarm robotic behaviors and current applications“. In: *Frontiers in Robotics and AI* 7 (2020), S. 36.
- [30] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus und Y. LeCun. „Overfeat: Integrated recognition, localization and detection using convolutional networks“. In: *arXiv preprint arXiv:1312.6229* (2013).
- [31] R. Shamshiri, C. Weltzien, I. Hameed, I. Yule, T. Grift, S. Balasundram, L. Pitonakova, D. Ahmad und G. Chowdhary. „Research and development in agricultural robotics: A perspective of digital farming“. In: (2018).
- [32] K. Simonyan und A. Zisserman. „Very deep convolutional networks for large-scale image recognition“. In: *arXiv preprint arXiv:1409.1556* (2014).
- [33] T. J. Soon. „QR code“. In: *synthesis journal* 2008 (2008), S. 59–78.
- [34] C. Szegedy, A. Toshev und D. Erhan. „Deep neural networks for object detection“. In: *Advances in neural information processing systems* 26 (2013).
- [35] J. Tang, G. Liu und Q. Pan. „A review on representative swarm intelligence algorithms for solving optimization problems: Applications and trends“. In: *IEEE/CAA Journal of Automatica Sinica* 8.10 (2021), S. 1627–1643.
- [36] W. von Thienen, D. Metzler, D.-H. Choe und V. Witte. „Pheromone communication in ants: a detailed analysis of concentration-dependent decisions in three species“. In: *Behavioral ecology and sociobiology* 68.10 (2014), S. 1611–1627.
- [37] I. Tsitsimpelis, C. J. Taylor, B. Lennox und M. J. Joyce. „A review of ground-based robotic systems for the characterization of nuclear environments“. In: *Progress in nuclear energy* 111 (2019), S. 109–124.

- [38] T. Tsujimura. *OLED display fundamentals and applications*. John Wiley & Sons, 2017.
- [39] J. Vartiainen, T. Kallonen und J. Ikonen. „Barcodes and mobile phones as part of logistic chain in construction industry“. In: *2008 16th International Conference on Software, Telecommunications and Computer Networks*. IEEE. 2008, S. 305–308.
- [40] E. Vincent und R. Lagani  re. „Detecting planar homographies in an image pair“. In: *ISPA 2001. Proceedings of the 2nd International Symposium on Image and Signal Processing and Analysis. In conjunction with 23rd International Conference on Information Technology Interfaces (IEEE Cat. IEEE*. 2001, S. 182–187.
- [41] T.-M. Wang, Y. Tao und H. Liu. „Current researches and future development trend of intelligent robot: A review“. In: *International Journal of Automation and Computing* 15.5 (2018), S. 525–546.
- [42] Z. Zeng, P.-J. Chen und A. A. Lew. „From high-touch to high-tech: COVID-19 drives robotics adoption“. In: *Tourism geographies* 22.3 (2020), S. 724–734.