



# UNIVERSITÄT LEIPZIG

Institut für Informatik  
Fakultät für Mathematik und Informatik  
Abteilung Datenbanken

## Methoden zur Überwindung der Simulation-to-Reality Gap

Bachelorarbeit

vorgelegt von:  
Merlin Flach

Matrikelnummer:  
3759809

Betreuer:  
Prof. Dr. Erhard Rahm  
Dr. Thomas Burghardt

© 2023

Dieses Werk einschließlich seiner Teile ist **urheberrechtlich geschützt**. Jede Verwertung außerhalb der engen Grenzen des Urheberrechtsgesetzes ist ohne Zustimmung des Autors unzulässig und strafbar. Das gilt insbesondere für Vervielfältigungen, Übersetzungen, Mikroverfilmungen sowie die Einspeicherung und Verarbeitung in elektronischen Systemen.

---

## **Abstract**

Fortschritte in der Leistungsstärke von Computern und im Bereich der künstlichen Intelligenz sorgen für ein wachsendes Interesse an autonomen Robotern. Um zeitliche und finanzielle Kosten zu sparen, werden diese häufig in Simulationen entwickelt. Dafür werden Neuronale Netze mithilfe eines Simulators entwickelt und schließlich in die reale Welt übertragen. Diese Arbeit beschäftigt sich mit der Überwindung von Problemen, die dabei auftreten. Am Beispiel eines autonomen Fahrzeuges, dessen Controller in einer Simulation entwickelt wurde, wurden die dabei auftretenden Komplikationen untersucht. Dazu gehören Schwierigkeiten bei der Bilderverarbeitung und der Umwandlung der vom Netz berechneten Ausgabedaten für Beschleunigung und Lenkwinkel in Werte, die vom Auto in der Realität nutzbar sind. Es wird darauf eingegangen, welche Methoden aus welchem Grund zur Bewältigung der Probleme angewandt wurden und warum es letztendlich nicht gelang, das gewünschte Verhalten des Roboters in der realen Welt zu erzielen.

# Inhaltsverzeichnis

<b>Abbildungsverzeichnis</b>	<b>III</b>
<b>Verzeichnis der Listings</b>	<b>IV</b>
<b>1. Einleitung</b>	<b>1</b>
1.1. Motivation . . . . .	1
1.2. Ziele der Arbeit . . . . .	1
<b>2. Grundlagen</b>	<b>2</b>
2.1. Vorwärtsgerichtete Neuronale Netze . . . . .	2
2.2. Evolutionäre Algorithmen . . . . .	3
2.3. Evolutionäre Robotik . . . . .	3
2.4. Computer Vision . . . . .	3
2.4.1. Vergleich zwischen RGB und HSV . . . . .	3
2.4.2. Konvertierung von RGB zu HSV . . . . .	4
2.4.3. Nach Farben suchen . . . . .	5
<b>3. Verwandte Arbeiten</b>	<b>6</b>
3.1. In Simulationen entwickelte Roboter . . . . .	6
3.2. Vor- und Nachteile der Nutzung von Simulationen . . . . .	6
3.3. Reality Gap beim autonomen Fahren . . . . .	6
3.4. Überwindung der Reality Gap . . . . .	7
<b>4. Forschungssetup</b>	<b>9</b>
4.1. Gegebenes neuronales Netz . . . . .	9
4.2. Vergleich zwischen Simulation und Realität . . . . .	9
4.2.1. Simulation . . . . .	9
4.2.2. Realität . . . . .	10
<b>5. Simulation-to-Reality Gap</b>	<b>13</b>
5.1. Überblick über die Probleme . . . . .	13
5.2. Bilder und Bildverarbeitung . . . . .	13
5.2.1. Rauschen . . . . .	14
5.2.2. Objekte im Hintergrund . . . . .	15
5.2.3. Vielfältigere Farben . . . . .	17
5.3. Nutzung des neuronalen Netzes . . . . .	19
5.3.1. Probleme beim neuronalen Netz . . . . .	19
5.3.2. Unterschiede bei den Eingabedaten . . . . .	20
5.3.3. Verarbeitung der Outputs . . . . .	27
5.3.3.1. Beschleunigung . . . . .	27
5.3.3.2. Lenkung . . . . .	27

<b>6. Evaluierung</b>	<b>31</b>
6.1. Ergebnisse der Computer Vision . . . . .	31
6.2. Ergebnisse bei der Übertragung des Neuronalen Netzes . . . . .	31
<b>7. Zusammenfassung und Ausblick</b>	<b>33</b>
7.1. Zusammenfassung . . . . .	33
7.2. Ausblick . . . . .	33
7.2.1. Computer Vision . . . . .	33
7.2.2. Anpassung der Simulation und des Neuronalen Netzes . . . . .	33
<b>Literatur</b>	<b>35</b>
<b>Selbständigkeitserklärung</b>	<b>38</b>

## **Abbildungsverzeichnis**

2.1. Vorwärtsgerichtetes neuronales Netz . . . . .	2
2.2. HSV-Farbraum Kegel . . . . .	4
2.3. Beispiel für die Erstellung einer binären Bildmaske . . . . .	5
3.1. Beispiel eines mit BlenderProc erstellten Bildes . . . . .	7
3.2. Kontrollarchitektur zur Überwachung von Roboteraktionen . . . . .	8
4.1. Simulation, in der das neuronale Netz trainiert wurde . . . . .	9
4.2. Arena, in der das Auto fährt . . . . .	10
4.3. Gebastelte Hindernisse . . . . .	11
4.4. NVIDIA JetBot . . . . .	11
5.1. Verarbeitungspipeline . . . . .	13
5.2. Vergleich zwischen aufgenommenem und verarbeiteten Bild mit und ohne Rauschen . . . . .	14
5.3. Vergleich zwischen den Methoden zur Eliminierung des Rauschen . . . . .	15
5.4. Raum, in dem die Arena steht . . . . .	15
5.5. Objekte im Hintergrund werden als Hindernis erkannt . . . . .	16
5.6. HSV-Werte einzelner Pixel aufgenommener Objekte . . . . .	16
5.7. Pappmauer, die am Rand der Arena installiert wurde . . . . .	17
5.8. Lichtsetup in der Realität . . . . .	17
5.9. Die Farbe rot wird auch im blauen Hindernis erkannt . . . . .	18
5.10. Bildmasken des roten Hindernisses mit hohem und niedrigem Farbwert . . . . .	19
5.11. Dunkle Stellen werden fälschlich als Bande wahrgenommen . . . . .	19
5.12. Ausgangspunkt für Test der Eingabedaten . . . . .	21
5.13. Veranschaulichung der Eingabedaten in Listing 5.2 . . . . .	21
5.14. Veranschaulichung der Eingabedaten in Listing 5.3 . . . . .	22
5.15. Veranschaulichung der Eingabedaten in Listing 5.4 . . . . .	22
5.16. Veranschaulichung der Eingabedaten in Listing 5.5 . . . . .	23
5.17. Veranschaulichung der Eingabedaten in Listing 5.6 . . . . .	23
5.18. Veranschaulichung der Eingabedaten in Listing 5.7 . . . . .	24
5.19. Veranschaulichung der Eingabedaten in Listing 5.8 . . . . .	25
5.20. Veranschaulichung der Eingabedaten in Listing 5.9 . . . . .	25
5.21. Veranschaulichung der Eingabedaten in Listing 5.10 . . . . .	26
5.22. Objekte im Hintergrund sind wieder erkennbar, wenn man die Position der Kamera an die Simulation anpasst . . . . .	26
5.23. Wellen auf der Oberfläche der Arena . . . . .	28
5.24. Gemessene Winkel beim Testen der Lenkung . . . . .	29

## **Verzeichnis der Listings**

5.1. Eingabe und Ausgabe des Ausgangsbildes . . . . .	20
5.2. Eingabe und Ausgabe ohne Breite und y-Koordinate . . . . .	21
5.3. Eingabe und Ausgabe ohne Bande . . . . .	21
5.4. Eingabe und Ausgabe bei begrenzter Anzahl an roten und blauen Hindernissen . . . . .	22
5.5. Eingabe und Ausgabe bei begrenzter Anzahl an roten und blauen Hindernissen ohne Bande . . . . .	22
5.6. Eingabe und Ausgabe, wenn ein rotes Hindernis entfernt wird . . . . .	23
5.7. Eingabe und Ausgabe, wenn ein blaues Hindernis entfernt wird . . . . .	23
5.8. Eingabe und Ausgabe bei höherem letzten roten Hindernis . . . . .	24
5.9. Eingabe und Ausgabe bei höherem ersten blauen Hindernis . . . . .	24
5.10. Eingabe und Ausgabe mit manipulierten Höhen . . . . .	25

# 1. Einleitung

## 1.1. Motivation

Immer häufiger werden autonome Roboter in Simulationen entwickelt, vor allem im Bereich der Evolutionären Robotik. Dafür gibt es mehrere Gründe. Es ist sicherer, billiger und zuverlässiger, als in der Realität [1]. Beim Übertragen der Robotersysteme in die Realität, kommt es jedoch häufig zu Problemen. Diese werden der Simulation-to-Reality Gap (Reality Gap) zugeschrieben. Sie bezieht sich auf den Unterschied zwischen der Leistung von Robotern, die in einer Simulation entwickelt wurden, und ihrer Leistung in der realen Welt. Diese Diskrepanz kann durch verschiedene Faktoren verursacht werden. In einer Vorarbeit<sup>1</sup> wurde ein autonomes Robotersystem innerhalb einer Simulation entwickelt. Dabei handelt es sich um ein Fahrzeug, das mit einer Kamera ausgestattet ist. Dieses kann in der Simulation innerhalb einer Arena autonom von einer Seite zur anderen fahren und dabei farbige Hindernisse vor sich wahrnehmen. Es ist in der Lage diese, abhängig von deren Farbe, zu umfahren. Der entwickelte Controller soll nun auf einen echten Roboter übertragen werden.

## 1.2. Ziele der Arbeit

Ziel ist es, in der Realität ein Modellauto zu konstruieren, welches mit einer Kamera den Bereich vor sich aufnehmen kann. Anschließend soll der in der Vorarbeit entstandene Controller auf dieses Fahrzeug übertragen werden. Dabei ist zu untersuchen, welche Probleme aufgrund der Reality Gap auftreten und worauf diese zurückzuführen sind. Daraufhin werden Methoden angewandt und vorgeschlagen, um die Probleme zu lösen und somit die Reality Gap bei diesem Fall zu überwinden.

---

<sup>1</sup>Bachelorarbeit von J. König

## 2. Grundlagen

### 2.1. Vorwärtsgerichtete Neuronale Netze

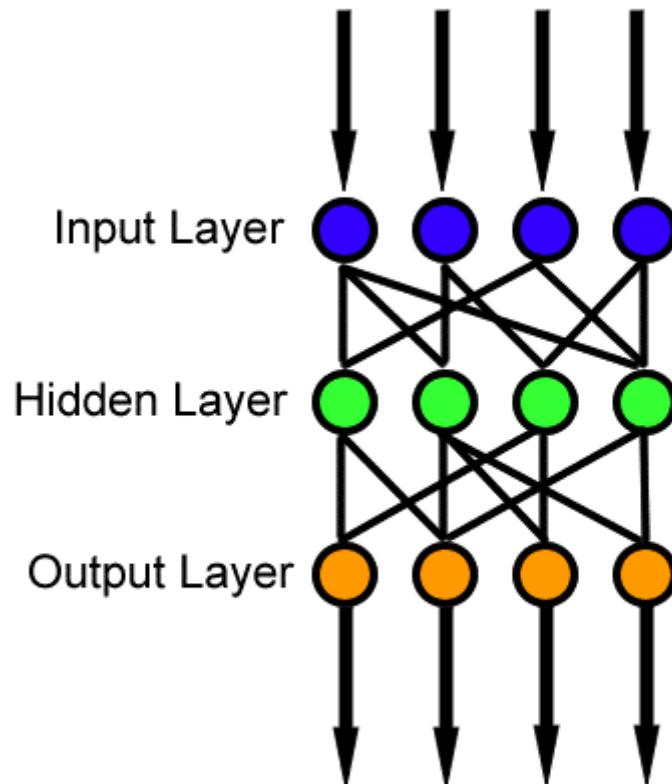


Abbildung 2.1.: Vorwärtsgerichtetes neuronales Netz, Quelle: Wikipedia<sup>2</sup>

Neuronale Netze sind Algorithmen, die im Bereich der künstlichen Intelligenz und des maschinellen Lernens genutzt werden. Sie sind dem menschlichen Gehirn nachempfunden und setzen sich aus drei Bereichen zusammen. Diese sind die Eingabeschicht (Input Layer), die verborgene Schicht (Hidden Layer) und die Ausgabeschicht (Output Layer) (siehe Abbildung 2.1). Die Anzahl der verborgenen Schichten ist je nach Netz unterschiedlich. Man bezeichnet ein neuronales Netz als vorwärtsgerichtet, wenn jede dazugehörige Schicht ausschließlich mit der nächst höheren verbunden ist. Vorwärtsgerichtete neuronale Netze ermöglichen die Lösung nicht-linearer Klassifizierungs- und Regressionsprobleme [2]. In der Eingabeschicht werden Informationen entgegengenommen. Diese werden über die versteckten Schichten geleitet, bis sie schließlich in der Ausgabeschicht landen. Die Daten in dieser Schicht stellen das berechnete Ergebnis dar. Das Trainieren eines Neuronalen Netzes kann als Optimierungsproblem betrachtet werden. Dabei besteht das Ziel darin, die für eine Aufgabe passenden Wichtungen und Verbindungen zwischen den jeweiligen Schichten zu finden.

<sup>2</sup>[https://en.wikipedia.org/wiki/File:Feed\\_forward\\_neural\\_net.gif](https://en.wikipedia.org/wiki/File:Feed_forward_neural_net.gif)

## 2.2. Evolutionäre Algorithmen

Evolutionärer Algorithmus ist ein Oberbegriff für populationsbasierte, stochastische und direkte Suchalgorithmen, die die natürliche Evolution nachahmen [3]. Diese Algorithmen sind vorteilhaft für das Trainieren neuronaler Netze. Bei bekannten Lernansätzen, wie beispielsweise der Backpropagation, werden häufig ein lokales Minimum statt des gesuchten globalen gefunden. Evolutionäre Algorithmen bieten aufgrund der stochastischen Natur eine bessere Möglichkeit dieses Problem zu vermeiden [4].

## 2.3. Evolutionäre Robotik

In der Robotik werden Maschinenbau, Elektrotechnik und Informatik kombiniert, um Roboter zu entwerfen und zu konstruieren. Einen Roboter beschreibt man als autonom, wenn er in der Lage ist ohne menschliches Eingreifen selbstständig zu Handeln und Aufgaben zu erledigen. Die evolutionäre Robotik befasst sich mit der automatischen Entwicklung autonomer Roboter mithilfe des darwinistischen Prinzips der natürlichen Selektion. Dafür werden evolutionäre Algorithmen genutzt. Oft ist das Ziel, das Verhalten des Robotersystems hinsichtlich einer Eigenschaft, wie zum Beispiel eine schnellstmögliche Fortbewegung, zu optimieren. In vielen Fällen entdecken die Lernalgorithmen Lösungen für gegebene Probleme, die nicht offensichtlich sind. Das tritt vor allem dann auf, wenn die Roboter nicht intuitiv für eine menschliche Steuerung sind [5].

## 2.4. Computer Vision

Um die gesuchten Hindernisse zu erkennen, müssen den aufgenommenen Bildern die dafür relevanten Informationen entnommen werden. Dies kann automatisch mithilfe der Computer Vision umgesetzt werden. In [6] wird die Computer Vision als Prozess definiert, bei dem Computer eingesetzt werden, um aus Bildern nützliche Informationen über die physische Welt zu extrahieren, einschließlich aussagekräftiger Beschreibungen von physischen Objekten. Im Rahmen dieser Forschung ist es die Aufgabe der Computer Vision, Objekte mit bestimmten Farben auf dem Bild zu lokalisieren und deren Größe zu bestimmen.

### 2.4.1. Vergleich zwischen RGB und HSV

Bilder werden in den meisten Fällen im RGB-Farbraum aufgenommen. Das bedeutet, dass jeder Pixel eines Bildes als Tupel ( $R, G, B$ ) dargestellt werden kann. Dabei kann  $R$ ,  $G$  und  $B$  jeweils einen Wert zwischen 0 und 255 haben, der den Anteil der jeweiligen Farbe (Rot, Grün oder Blau) beschreibt. Wie in [7], [8], [9] beschrieben, kommt es durch eine Änderung der Beleuchtung oder Schatten zu teilweise starken Farbabweichungen. Folglich ändern sich die Werte der Farbtupel stark. Daher ist der RGB-Farbraum nicht geeignet, um auf diese Weise nach Objekten auf Bildern zu suchen. Stattdessen werden für die Bildverarbeitung die RGB-Daten aus dem Bild zu HSV konvertiert [8]. Veränderungen der Lichtverhältnisse haben einen geringeren Einfluss im HSV-Farbspektrum. Wie

bei RGB, können im HSV-Format Pixel ebenso als Tupel ( $H, S, V$ ) dargestellt werden.  $H$  (hue) repräsentiert die Farbe und reicht von 0 bis 179.  $S$  (saturation) entspricht der Intensität der Farbe und hat einen Wert zwischen 0 und 255, wobei 255 die stärkste Farbe enthält.  $V$  (value) beschreibt die Helligkeit der Farbe und reicht von 0 bis 255, wobei 255 am hellsten ist (siehe 2.2).

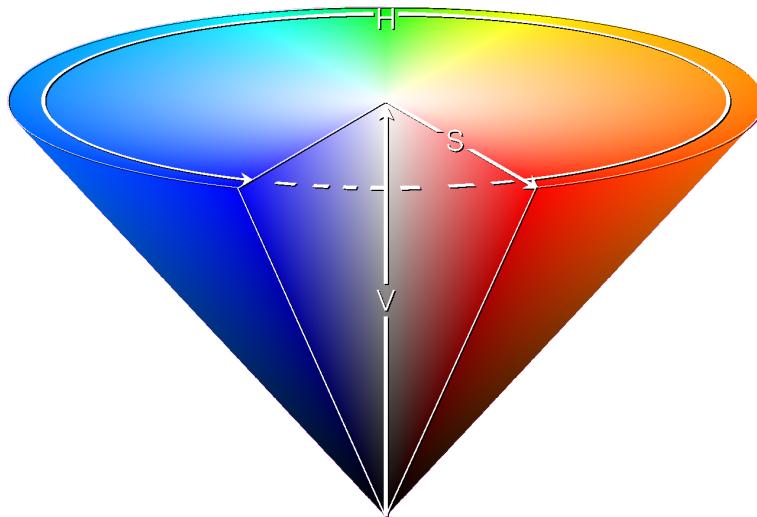


Abbildung 2.2.: HSV-Farbraum Kegel, Quelle: Wikipedia<sup>3</sup>

#### 2.4.2. Konvertierung von RGB zu HSV

Um die RGB-Daten zu HSV zu konvertieren, müssen erst R, G und B zu Gleitkommazahlen umgewandelt und skaliert werden, dass sie in das Intervall zwischen 0 und 1 passen.

$$R' = \frac{R}{255}, \quad G' = \frac{G}{255}, \quad B' = \frac{B}{255} \quad (2.1)$$

$H$ ,  $S$  und  $V$  entsprechen dann jeweils folgenden Berechnungen:

$$V = \max(R', G', B') \quad (2.2)$$

$$S = \begin{cases} \frac{V - \min(R', G', B')}{V} & \text{falls } V \neq 0 \\ 0 & \text{sonst} \end{cases} \quad (2.3)$$

$$H = \begin{cases} \frac{60 \cdot (G' - B')}{V - \min(R', G', B')} & \text{falls } V = R' \\ 120 + \frac{60 \cdot (B' - R')}{V - \min(R', G', B')} & \text{falls } V = G' \\ 240 + \frac{60 \cdot (R' - G')}{V - \min(R', G', B')} & \text{falls } V = B' \\ 0 & \text{falls } R' = G' = B' \end{cases} \quad (2.4)$$

---

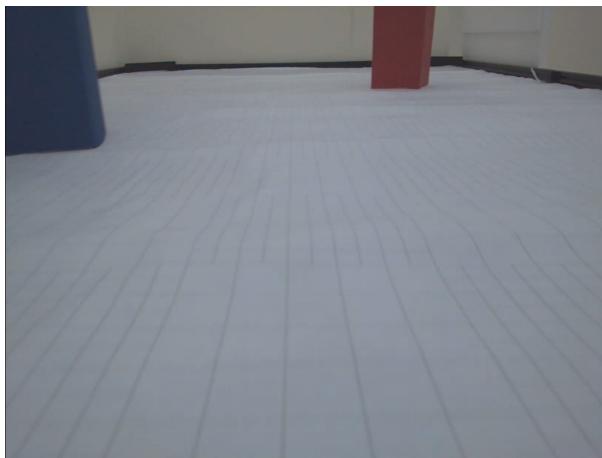
<sup>3</sup>[https://de.wikipedia.org/wiki/HSV-Farbraum#/media/Datei:HSV\\_cone.png](https://de.wikipedia.org/wiki/HSV-Farbraum#/media/Datei:HSV_cone.png)

Für den Fall, dass  $H < 0$  gilt, setzt man  $H = H + 360$ . Anschließend werden die Werte noch an den gewünschten Datentypen angepasst.

$$H = \frac{H}{2}, S = 255 \cdot S, V = 255 \cdot V \quad (2.5)$$

### 2.4.3. Nach Farben suchen

Nach der Konvertierung in den HSV-Farbraum kann nach Farben gesucht werden. Dafür werden binäre Bilder genutzt. Diese beinhalten nur zwei Werte: Schwarz (0) und Weiß (1) (siehe 2.3b). Durch diese Darstellung eines Bildes werden unwichtige Informationen entfernt, was die Analyse der



(a) Aufgenommenes Bild



(b) Binäres Bild mit gefundener Farbe Rot

Abbildung 2.3.: Beispiel für die Erstellung einer binären Bildmaske

relevanten Daten erleichtert. Um ein solches binäres Bild zu erstellen, müssen Schwellenwerte für den abgesuchten HSV-Bereich der jeweils gesuchten Farbe festgelegt werden. Aus dem aufgenommenen Bild wird dann eine Bildmaske mit den gesuchten HSV-Werten berechnet, die das binäre Bild repräsentiert [8], [9]. Diese kann man als Matrix aller Pixel im Bild betrachten, in denen nur die Werte 0 und 1 vorkommen. Dabei stellt der Wert 1 dar, dass die Farbe in dem jeweiligen Pixel gefunden wurde. Daraus kann man schließlich die Positionen der Bereiche mit der gesuchten Farbe auf der Aufnahme entnehmen.

## 3. Verwandte Arbeiten

### 3.1. In Simulationen entwickelte Roboter

Schon seit mehreren Jahren werden Simulationen genutzt, um autonome Robotersysteme zu entwickeln und diese zu testen. In [10] wurde beispielsweise ein Fahrzeug innerhalb einer Simulation trainiert, dass mit Infrarot- und Umgebungslichtsensoren ausgestattet ist. Anschließend wurde untersucht, wie effektiv es Hindernisse umfahren und Licht auffinden konnte. Dabei konnten in der Realität nahezu identische Verhalten erzielt werden. Weiterhin wurden in [11] neuronale Netze mithilfe eines Simulators entwickelt, die auf einen achtbeinigen Roboter übertragen wurden. Dieser konnte sich erfolgreich frei umherbewegen und dabei herumliegenden Objekten mithilfe einer Sensorwahrnehmung ausweichen oder bei Kontakt von diesen zurückweichen. Lee et al. [12] gelang es, einem blinden Roboter mit vier Beinen zu ermöglichen, autonom herausfordernde natürliche Umgebungen zu überwinden. Der dafür nötige Controller wurde mithilfe von Reinforcement Learning in einer simulierten Umgebung entwickelt.

### 3.2. Vor- und Nachteile der Nutzung von Simulationen

Allgemein bieten Simulatoren viele Vorteile für die Entwicklung von Robotersystemen. Sie können das trainieren neuronaler Netze beschleunigen und bieten mehr Sicherheit vor der Beschädigung von Hardware [13], [14]. Außerdem wird keine menschliches Überwachen oder Eingreifen benötigt [15]. Besonders im Bereich der evolutionären Robotik ist es deshalb notwendig, Simulationen zur Entwicklung Neuronaler Netze zu nutzen. Es kommt jedoch häufig dazu, dass in Simulationen entwickelte Controller völlig ineffizient werden, sobald sie auf echte, physische Roboter übertragen werden [16]. Dieses Problem wird als Reality Gap bezeichnet. Gründe dafür sind beispielsweise in falschen Mengen berücksichtigtes Rauschen [10] oder schlecht modellierte Phänomene, die dazu führen, dass unrealistische Verhaltensweisen erlernt werden [16]. Zagal et al. [17] erklären, dass es davon kommt, dass Simulationen aus willkürlichen Offline-Designs entwickelt werden und deshalb zu stark von der Realität abweichen. In [18] wurde untersucht, ob es deshalb notwendig ist die Komplexität der Simulation an die der realen Welt anzupassen. Dabei kam man zu dem Ergebnis, dass es auch ohne Unterschiede in der Komplexität zu schlechterer Performance nach der Übertragung der künstlichen Intelligenz kommt.

### 3.3. Reality Gap beim autonomen Fahren

Im Fall des autonomen Fahren, stellt häufig die Erkennung von Objekten mithilfe von Kameras oder Sensoren ein Problem dar [19]. Da unzählige Testfahrten stattfinden müssen, um die Sicherheit zu garantieren, werden in der Autoindustrie Simulationssoftwares genutzt, um diese Tests durchzuführen [20]. Diese bieten die Möglichkeit kontrollierbare Konditionen für die Tests zu schaffen. In [20],

[21] werden Methoden vorgestellt, mit denen man die Reality Gap bei einer kamera- oder sensorbasierten Hinderniserkennung messen kann. Dafür werden Daten aus einem Ground-Truth-Datensatz mit den in der Simulation gewonnenen Daten verglichen.

### 3.4. Überwindung der Reality Gap

Es gibt zahlreiche Forschungen, die sich mit der Überwindung der Reality Gap befassen. Tremblay et al. [22] beschäftigten sich mit der Bewältigung der Reality Gap bei der Erkennung von Objekten auf Bildern. Dazu wurde die Technik der Domain Randomization genutzt, um der Variabilität realer Daten gerecht zu werden. Das heißt, innerhalb der Simulation wurden verschiedene Parameter, wie Licht, Objekttexturen, etc., zufällig und auf unrealistische Weise gesetzt, damit das neuronale Netz erlernt, was die wichtigsten Elemente eines gesuchten Objektes sind. In [23] wurde stattdessen BlenderProc<sup>4</sup> genutzt, um fotorealistische Bilder aus prozedural generierten 3D-Szenen zu rendern (vgl. Abbildung 3.1). Diese wurden zum Trainieren datenintensiver Deep-Learning-Modelle verwendet. Dadurch konnte die Reality Gap zwischen synthetischen und realen Tests bei der Bild-



Abbildung 3.1.: Beispiel eines mit BlenderProc erstellten Bildes, Quelle: [23]

verarbeitung reduziert werden. Zagal und Ruiz-del-Solar [17] schlagen hingegen vor, die Reality Gap zu überwinden, indem umweltangepasste Simulationen verwendet werden. Diese können durch die

---

<sup>4</sup><https://github.com/DLR-RM/BlenderProc>

Ko-Evolution von Roboterverhalten und Simulator erreicht werden. Es wird ein Algorithmus präsentiert, der die Unterschiede im Verhalten eines Roboters in der realen Welt und in der Simulation ausnutzt, um den Simulator zu adaptieren. Ein weiterer Lösungsvorschlag zur Überwindung der Reality Gap ist der Ansatz der Übertragbarkeit, welcher in [16] vorgestellt wird. Dabei sollen die Fitness des Roboters und die Übertragbarkeit des Controllers mithilfe eines Pareto-basierten, multikriteriellen evolutionären Algorithmus optimiert werden. Beide Eigenschaften werden durch ein Maß für die Ungleichheit zwischen Simulation und Realität approximiert. In [24] wurde ebenso die verwendete Simulation angepasst, um eine höhere Übertragbarkeit der entwickelten Controller und somit bessere Ergebnisse hinsichtlich der Überwindung der Reality Gap zu erzielen. Dafür wurden Parameter der Physik-Engine der Simulation, wie Reibung und maximale Antriebsgeschwindigkeit des Roboters, optimiert. Hartland und Bredeche [25] arbeiteten hingegen an einer Kontrollarchitektur, die die Fehler eines bereits in die Realität übertragenen Controllers ausgleichen soll. Um dies zu erreichen, wurden ein Anticipations- und ein Korrekturmodul entwickelt, welche die Ausgaben des Controllers überwachen und korrigieren (siehe Abbildung 3.2). Das Anticipationsmodul besitzt

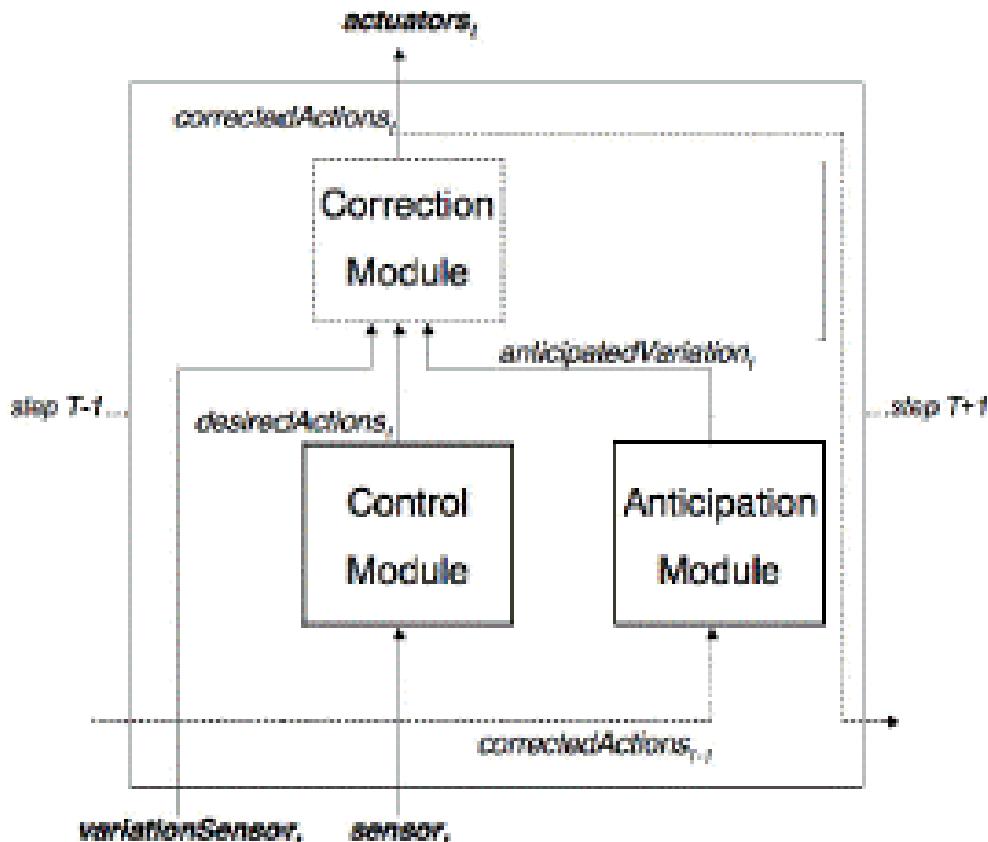


Abbildung 3.2.: Kontrollarchitektur zur Überwachung von Roboteraktionen, Quelle: [25]

ebenso künstliche Intelligenz und muss demnach trainiert werden. Es schlägt in Echtzeit Änderungen an den vom Controller berechneten Ausgabedaten vor. Das Korrekturmodul verarbeitet diese Änderungen und passt gegebenenfalls die Daten an, bevor sie an den Motortreiber weitergegeben werden.

## 4. Forschungssetup

### 4.1. Gegebenes neuronales Netz

In der Vorarbeit zu dieser Forschung wurden mehrere vorwärtsgerichtete neuronale Netze mithilfe von evolutionären Algorithmen entwickelt. Dies geschah innerhalb einer Simulation. Hier wird nur das Netz betrachtet, für welches die beste Performance dokumentiert wurde. Es wurde über 100 Generationen mit je einer Population von 15 Fahrzeugen trainiert. Die Eingabedaten für das neuronale Netz sind die Positionen der Hindernisse, welche mithilfe der Computer Vision berechnet werden (siehe Abschnitt 2.4). Diese haben die Form eines Tupels  $(x, y, w, h)$ , wobei  $(x, y)$  die Koordinaten der oberen linken Ecke und  $(w, h)$  die Breite und Höhe des gefundenen Rechtecks darstellen. Aus einer Liste aller erkannten Hindernisse berechnet das Netz die Beschleunigung und den Lenkwinkel, mit denen das Fahrzeug fahren soll. Die ausgegebenen Werte befinden sich jeweils im Intervall zwischen -1 und 1. Für die Beschleunigung heißt ein positiver Wert, dass das Auto Gas geben soll. Bei einem negativen Wert soll es abbremsen, beziehungsweise rückwärts fahren. Die Ausgabe für den Lenkwinkel lässt sich in einen Winkel zwischen  $0^\circ$  und  $180^\circ$  übersetzen. Ist der Wert negativ, soll nach links gelenkt werden. Wenn der Wert positiv ist, soll stattdessen nach rechts gelenkt werden.

### 4.2. Vergleich zwischen Simulation und Realität

#### 4.2.1. Simulation

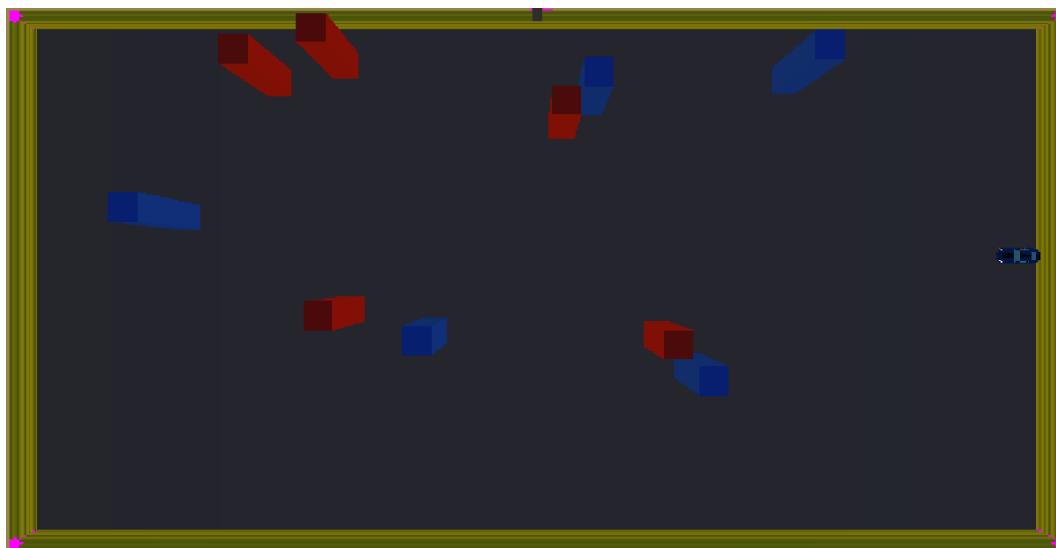


Abbildung 4.1.: Simulation, in der das neuronale Netz trainiert wurde

Die Simulation, in der das neuronale Netz trainiert wurde, wurde in Unity<sup>5</sup> entwickelt. In ihr wurde eine Arena aus der realen Welt nachgebaut. Diese wurde von einem ambienten Licht beleuchtet,

---

<sup>5</sup><https://unity.com/de>

sodass an allen Stellen die gleichen Lichtverhältnisse bestehen. Zum Trainieren des Netzes wurden innerhalb der simulierten Arena jeweils fünf rote und blaue Hindernisse mit identischen Formen und Maßen an zufälligen Positionen aufgestellt (vgl. Abbildung 4.1). Weiterhin befand sich eine gelbe Bande am Rand der Arena, die ebenfalls ein zu berücksichtigendes Hindernis darstellt. Ein kleines Auto musste sich autonom, so schnell wie möglich, von einer Seite der Arena zur Anderen navigieren. Zusätzlich sollten die roten Objekte links und die blauen rechts umfahren werden. Dafür wurde es mit einer Kamera ausgestattet, die mit einem Blickfeld von  $60^\circ$  den Bereich vor dem Auto aufnimmt. Die aufgenommenen Bilder haben eine Auflösung von 240x135 Pixel. Im Falle, dass es das Fahrzeug nicht schaffte ein Hindernis richtig zu umfahren, wurden ihm Strafsekunden angerechnet. Dabei konnte das Auto wie ein herkömmliches beschleunigen, bremsen, rückwärts fahren und um einen bestimmten Winkel lenken. Genaue Maße zu den verwendeten Hindernissen, oder der Höhe der Kamera über der Oberfläche, können nicht aus der Simulation ermittelt werden. Das liegt daran, dass die Abmessungen aller Objekte lediglich mit Positionen und Skalierungen angegeben werden. Zusätzlich wurden die Objekte nicht relativ zur Arena positioniert, sondern frei in einem Raum. Dadurch wurde die Bestimmung genauer Maße weiter erschwert.

##### 4.2.2. Realität

In der Realität wird an einer Arena gearbeitet, die zwei Meter lang und ein Meter breit ist (siehe Abbildung 4.2). Die Oberfläche ist mit einer Plane bedeckt, die einige Unebenheiten aufweist. Die



Abbildung 4.2.: Arena, in der das Auto fährt

Arena befindet sich in einem geschlossenen Raum und wird von mehreren Deckenlampen direkt beleuchtet. Demnach gibt es Unregelmäßigkeiten bei der Helligkeit an verschiedenen Stellen innerhalb in der Arena. Um die Hindernisse aus der Simulation nachzustellen, wurden Teile der Pappe eines Kartons zu einem Quader geformt und schließlich mit farbigem Tonkarton beklebt (siehe Abbildung 4.3). Während der Forschung wurden jeweils ein rotes und blaues Hindernis genutzt. Diese sind unterschiedlich breit und hoch. Das Auto, auf welches das neuronale Netz übertragen

#### 4. Forschungssetup

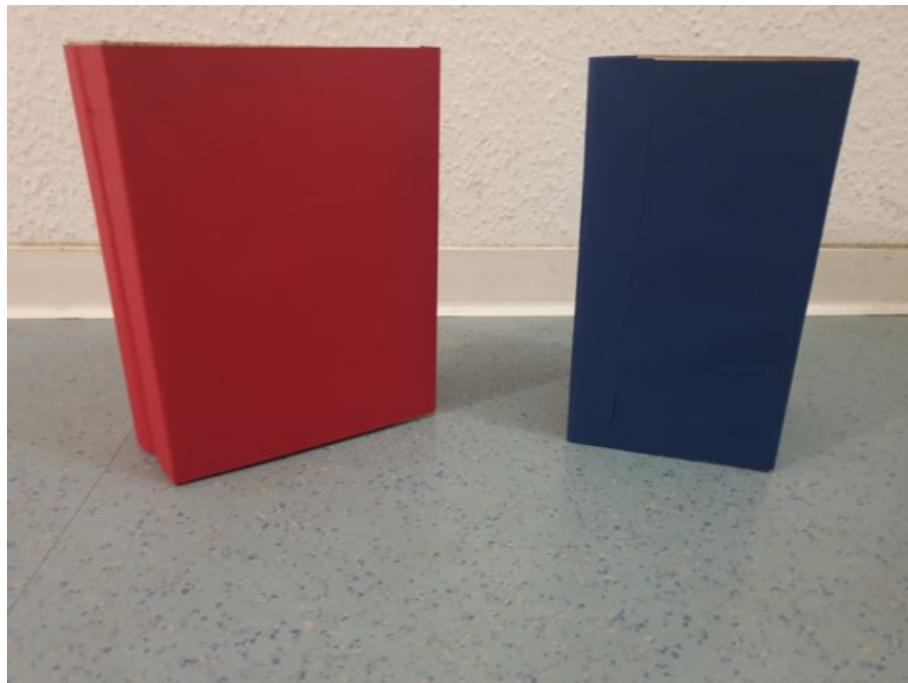


Abbildung 4.3.: Gebastelte Hindernisse

wird, ist ein NVIDIA<sup>6</sup> JetBot (siehe Abbildung 4.4). Es wurde sich für dieses Auto entschieden, da die dafür benötigte Recheneinheit bereits zur Verfügung stand und ein möglichst geringes Budget aufgebracht werden sollte. Zur Konstruktion und Einrichtung des JetBot wurde bis auf wenige Ab-

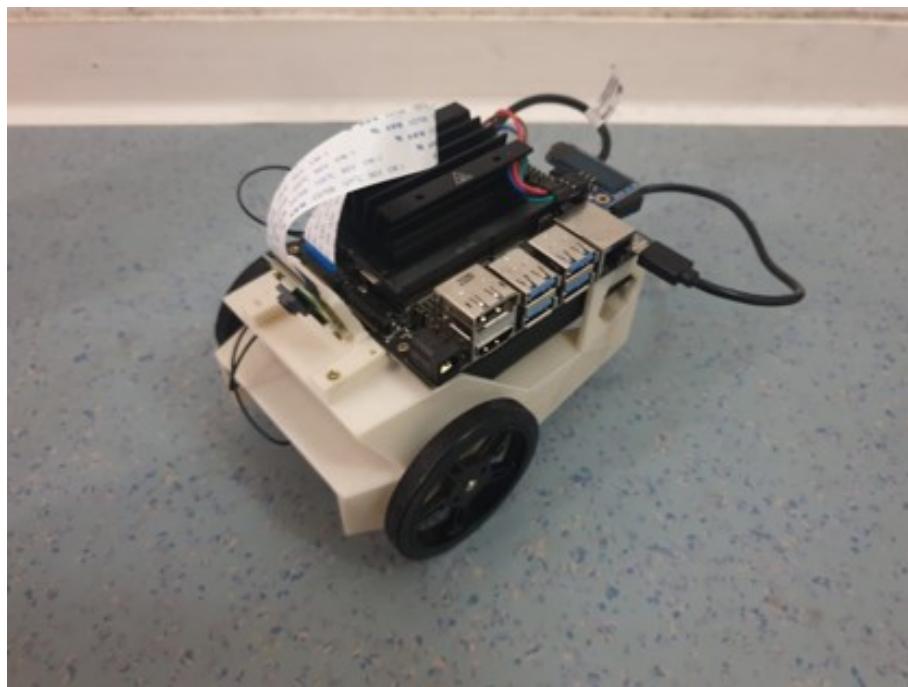


Abbildung 4.4.: NVIDIA JetBot

weichungen die Anleitung<sup>7</sup> des Entwicklerteams befolgt. Das Auto wird von einem Mini-Computer betrieben. Dabei handelt es sich um einen NVIDIA Jetson Nano. Der JetBot ist 11cm breit und

---

<sup>6</sup><https://www.nvidia.com>

<sup>7</sup><https://jetbot.org/master/>

#### *4. Forschungssetup*

---

14,3 cm lang. Anders als in der Anleitung wurde für die Aufnahme von Bildern ein Raspberry Pi Kamera-Modul V2 installiert, das wie die Kamera in der Simulation einen Blickwinkel von 60° hat. Das Objektiv der Kamera befindet sich ca. 8,7cm über der Oberfläche und ist um 20° nach unten geneigt. Aufgenommene Bilder haben eine Auflösung von 816x616 Pixel. Der JetBot besitzt zwei Räder, die von zwei unabhängigen Motoren gedreht werden können. Die Räder können im Gegensatz zu dem Auto in der Simulation nicht nach rechts oder links rotiert werden. Stattdessen kann eine Lenkung erzielt werden, indem die Räder unterschiedlich schnell drehen. Dreht das rechte Rad schneller, lenkt das Auto nach links. Wenn hingegen das linke Rad schneller dreht, wird nach rechts gelenkt. Ein Motortreiber wandelt die vom Jetson Nano übergebenen Werte für die Geschwindigkeit der Räder in Signale für die dazugehörigen Motoren um. Diese übermittelt er mithilfe eines On-Board-Kommunikationsprotokolles an die Motoren. Weiterhin wurde eine andere Batterie verwendet, als in der Anleitung, da die dort vorgeschlagene nicht verfügbar war. Stattdessen wurde die Powerbank der Marke INUI mit der Modellnummer BI-B41 zum Betreiben des NVIDIA Jetson Nano und des Motortreibers genutzt.

## 5. Simulation-to-Reality Gap

Um den JetBot, wie in der Simulation, autonom durch die Arena fahren zu lassen, muss der folgende Prozess in einer Schleife ausgeführt werden. Die Kamera, die am Fahrzeug befestigt ist, nimmt ein Bild auf. Mit Hilfe der Computer Vision werden aus diesem Bild automatisch die jeweils relevanten Informationen über die sichtbaren Hindernisse extrahiert (vgl. Abschnitt 2.4). Die dabei gesammelten Daten stellen die Eingabedaten für das neuronale Netz dar. Daraus berechnet das Netz Werte für die Beschleunigung und den Lenkwinkel, mit denen das Fahrzeug fahren soll. Diese Ausgabedaten müssen in passender Form an den Motortreiber weitergegeben werden. Er wandelt die erhaltenen Werte in Signale zum Antrieb der Motoren um, die die Räder zum Drehen bringen. Abbildung 5.1 zeigt rot markierte Bereiche. Das sind die Teile der Verarbeitungsschleife, die anfällig für das Aufkommen von Problemen durch die Reality Gap sind. Im folgenden Abschnitt werden die in dieser Forschung aufgetretenen Probleme erläutert und Lösungsmethoden vorgeschlagen.



Abbildung 5.1.: Verarbeitungspipeline

### 5.1. Überblick über die Probleme

Es gibt mehrere Schwierigkeiten, um mithilfe der Computer Vision eine gute Objekterkennung zu erzielen. Anders als in der Simulation, gibt es in der Realität Rauschen auf den aufgenommenen Bildern. Weiterhin führen die direkten Lichtquellen dazu, dass die aufgestellten Hindernisse Schatten werfen. Das sorgt dafür, dass Farben in unterschiedlich hellen Tönen auftreten. Demzufolge muss ein breiteres Farbspektrum abgesucht werden, um eine Farbe verlässlich finden zu können. Die hat als Folge, dass farbähnliche Objekte fälschlich als Hindernisse erkannt werden können. Erst wenn die Hinderniserkennung gut funktioniert, können sinnvolle Eingabedaten für das neuronale Netz aus den Bildern extrahiert werden. Da die Position, Ausrichtung und Auflösung der Kamera und die Maße der Hindernisse zu denen in der Simulation abweichen, weichen ebenso die aus den Bildern extrahierten Eingabedaten ab. Das hat einen direkten Einfluss auf die Berechnung der Ausgaben für Beschleunigung und Lenkung des Fahrzeugs. Zusätzlich können die Ausgabedaten des neuronalen Netzes nicht in der zurückgegebenen Form verwendet werden, da der JetBot grundlegend anders funktioniert, als das Auto in der Simulation. Demnach müssen sie vorher so umgewandelt werden, dass sie auf den JetBot übertragbar sind.

### 5.2. Bilder und Bildverarbeitung

Bilder in der Realität sind durch ihre zahllosen Details schwer zu simulieren. Deshalb stellt die Bildverarbeitung ein großes Hindernis für die Überwindung der Reality Gap dar. Aus diesem Grund

werden in modernen autonomen Fahrsystemen, zusätzlich zu Kameras, noch Radar-, Ultraschall- und LiDAR-Sensoren installiert [26]. In ähnlichen Forschungen, wie [10], wird gänzlich auf Kameras verzichtet, und stattdessen ausschließlich mit Sensoren gearbeitet. In dieser Forschung wurden zur Umsetzung der Computer Vision Funktionen von OpenCV genutzt. Dazu gehört `cvtColor()` für die Umwandlung der RGB-Farbwerke in HSV. Anschließend wurden die Funktion `inRange()` zum Suchen der Farben innerhalb der gewählten HSV-Schwellenwerte und `findContours()` für die Erzeugung der binären Bildmaske genutzt. Die Tupel der gefundenen Rechtecke konnten schlussendlich mithilfe von `boundingRect()` zurückgegeben werden.

### 5.2.1. Rauschen

Die Bilder, die das Fahrzeug in der Simulation aufnimmt, sind frei von Rauschen. Jeder Pixel eines Objekts hat dort die gleichen Farbwerte. Bei den Bildern, die der JetBot aufnimmt ist dies nicht der

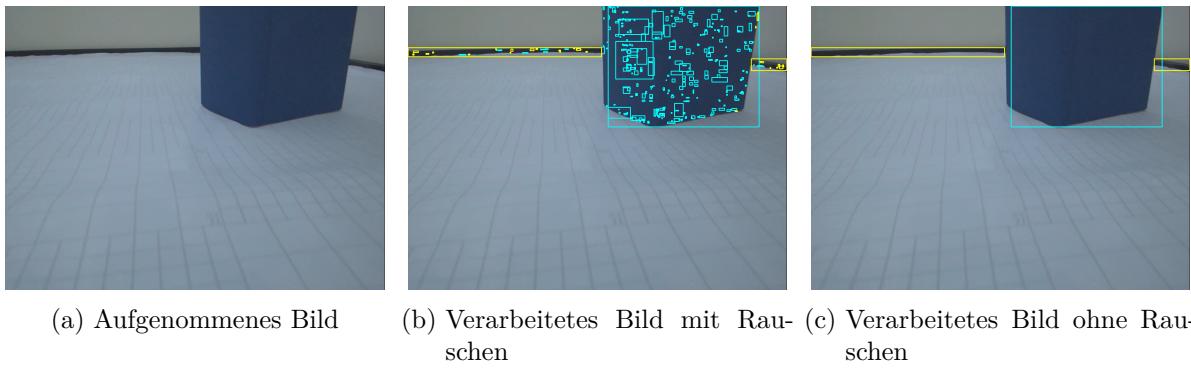
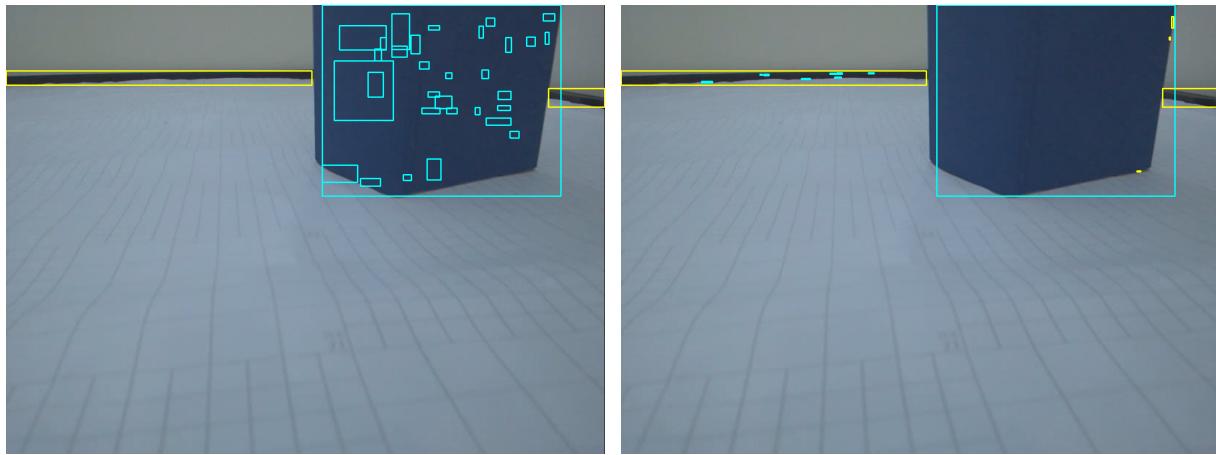


Abbildung 5.2.: Vergleich zwischen aufgenommenem und verarbeiteten Bild mit und ohne Rauschen

Fall. Man kann auf den Aufnahmen, innerhalb eines aufgestellten Hindernisses, auch Pixel mit abweichenden Farben auffinden. Dies hat zur Folge, dass bei der Verarbeitung der Bilder aufgrund von Rauschen mehrere kleine Hindernisse fälschlich erkannt werden (siehe Abbildung 5.2b). Die Computer Vision muss demnach so angepasst werden, dass die durch Rauschen irrtümlich erkannten Rechtecke während der Bildverarbeitung beseitigt werden. Zum Einen kann man festlegen, dass die gefundenen Rechtecke eine Mindestfläche haben müssen. In Abbildung 5.3a sieht man das Ergebnis, wenn man eine Mindestfläche von 100 Pixeln wählt. Dadurch können einige kleine Rechtecke ausgeschlossen werden. Dieser Ansatz ist nicht optimal, da einige der fälschlich erkannten Hindernisse als relativ groß wahrgenommen werden (vgl. Abbildung 5.3a). Wenn man eine zu hohe Mindestgröße der gesuchten Hindernisse festlegt, schließt man Objekte aus, die klein sind, oder klein erscheinen, weil ein Großteil davon über den Bildrand hinaus nicht zu sehen ist. Bessere Ergebnisse für die Eliminierung des Rauschens können erzielt werden, indem die erkannten Rechtecke gruppiert werden. Dabei sollen sie jeweils zu Rechtecken zusammengefasst werden, die dann den tatsächlich erkannten Objekten zugeordnet werden können. Dafür bietet sich die Nutzung der `groupRectangles()` Funktion von OpenCV an. Bei dieser Funktion kann ein Faktor definiert werden, der bestimmt, wie viel Ähnlichkeit die Rechtecke in Position und Größe haben müssen, um miteinander gruppiert zu werden. Je höher der Faktor ist, desto eher werden die Rechtecke gruppiert. In dieser Forschung wurde der Wert 0,2 gewählt. Der geringe Wert ist sinnvoll, da hauptsächlich das Rauschen innerhalb eines erkannten Objektes entfernt werden muss. Dabei überschneiden sich die Positionen der

## 5. Simulation-to-Reality Gap

Rechtecke, was für eine hohe Ähnlichkeit sorgt. In Abbildung 5.3b erkennt man, dass das Rauschen dadurch nahezu vollständig beseitigt wird. Es bleiben lediglich einige kleine Rechtecke übrig, die nicht richtig erkannt werden. Erst durch eine Kombination aus beiden Methoden werden alle durch das Rauschen irrtümlich erkannten Hindernisse beseitigt. Übrig bleiben die Rechtecke, die jeweils einem erkannten Objekt zugeordnet werden können (siehe Abbildung 5.2c).



(a) Mindestfläche für erkannte Rechtecke

(b) Gruppieren der erkannten Rechtecke

Abbildung 5.3.: Vergleich zwischen den Methoden zur Eliminierung des Rauschen

### 5.2.2. Objekte im Hintergrund

Die Arena, in welcher der JetBot fahren soll steht in einem geschlossenen Raum, der auch anderweitig genutzt wird (siehe Abbildung 5.4). Somit befinden sich auch Möbel und Objekte darin, die



Abbildung 5.4.: Raum, in dem die Arena steht

auf den aufgenommenen Bildern zu sehen sind. Einige dieser Objekte haben Farben, die ähnlich zu denen der Hindernisse sind, die der JetBot vermeiden soll. Häufig werden dunkle Elemente des Raumes als Bande erkannt. Das liegt daran, dass die Bande einen sehr großen Bereich an möglichen Farbwerten abdeckt (vgl. Abbildung 5.6). Allerdings wird einiges auch als blaues Hindernis erkannt

## 5. Simulation-to-Reality Gap

(siehe Abbildung 5.5). In Abbildung 5.6 stellen die roten, blauen und schwarzen Punkte je einen

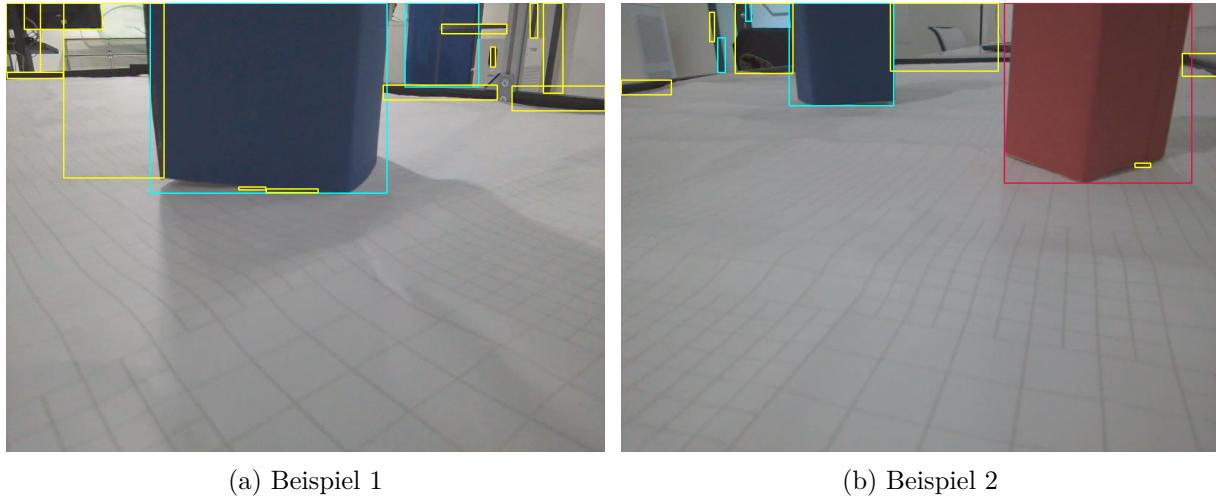


Abbildung 5.5.: Objekte im Hintergrund werden als Hindernis erkannt

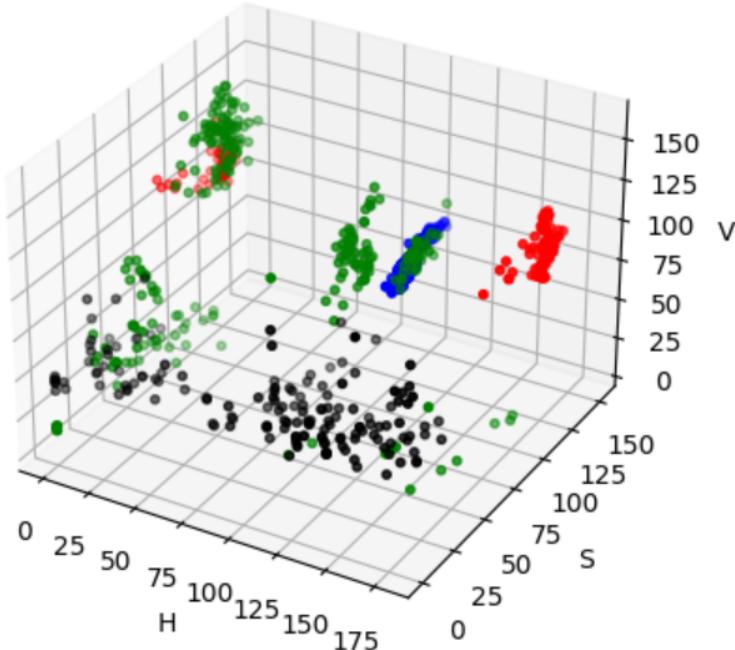


Abbildung 5.6.: HSV-Werte einzelner Pixel aufgenommener Objekte

zufällig ausgewählten Pixel innerhalb der dazugehörigen relevanten Hindernisse dar. Die grünen Punkte gehören zu farbähnlichen Mustern oder Objekten, die man im Hintergrund einiger Aufnahmen sehen kann. Dazu gehören zum Beispiel rot-braune Türen, verschiedene Monitore, Gestelle, oder blaue Aufdrucke auf den Möbeln. Besonders bei den blauen Pixeln und den roten Pixeln im niedrigen Farbwertbereich kann man erkennen, dass sich dichte Wolken mit Pixeln ähnlicher Farbe darum bilden. Diese gehören zu Objekten, die nicht als Hindernisse berücksichtigt werden sollen. Somit ist es nicht möglich, die gewählten HSV-Schwellenwerte für die Farberkennung so zu wählen, dass ausschließlich die aufgestellten Objekte in der Arena wahrgenommen werden. Demnach ist es notwendig, die im Raum befindlichen Objekte auf den Aufnahmen zu verstecken. Dies wird in dieser Forschung durch die Installation einer farbneutralen Mauer aus Pappe um die Arena erzielt

## 5. Simulation-to-Reality Gap

(siehe Abbildung 5.7). Nachteil daran ist, dass die Mauer einen Schatten wirft, der wiederum die Computer Vision beeinflusst. Dadurch müssen die HSV-Schwellenwerte für die Farbsuche angepasst werden, da die Hindernisse im Schatten eine dunklere Farbe haben. In Richtung der Wand musste keine Pappe aufgestellt werden, da das Weiß, beziehungsweise helle Grau, innerhalb des abgesuchten Farbbereiches nicht von der Computer Vision erkannt wird.

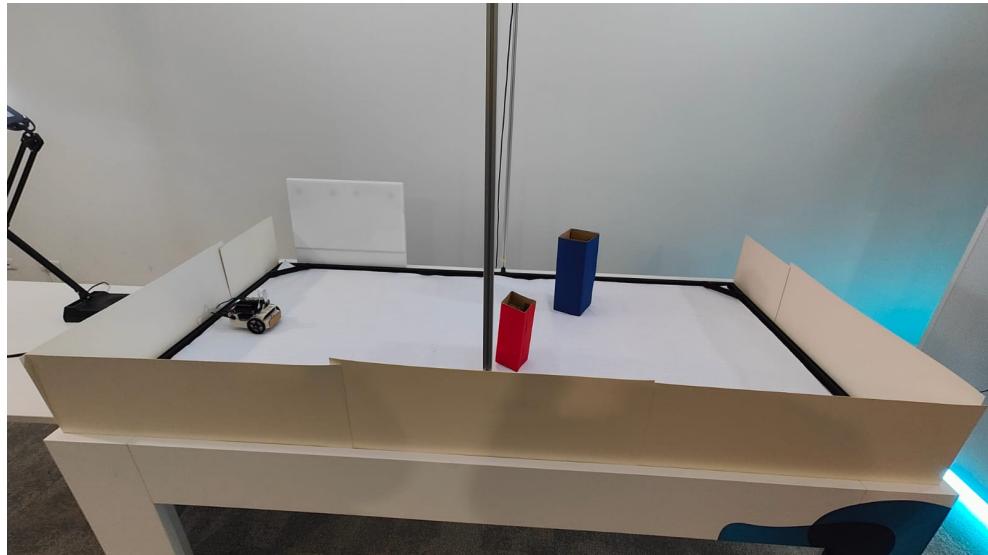


Abbildung 5.7.: Pappmauer, die am Rand der Arena installiert wurde

### 5.2.3. Vielfältigere Farben

In der Simulation gibt es keine direkte Lichtquelle, die auf die Arena scheint. Es gibt lediglich ein ambientes Licht, dass alles gleichmäßig erhellt. Somit entstehen auch keine Schatten oder anderweitig unregelmäßig helle oder dunkle Stellen (vgl. Abbildung 4.1). In der Realität bestehen jedoch keine derartigen Lichtverhältnisse (vgl. Abbildung 5.8). Es strahlen mehrere Deckenlampen

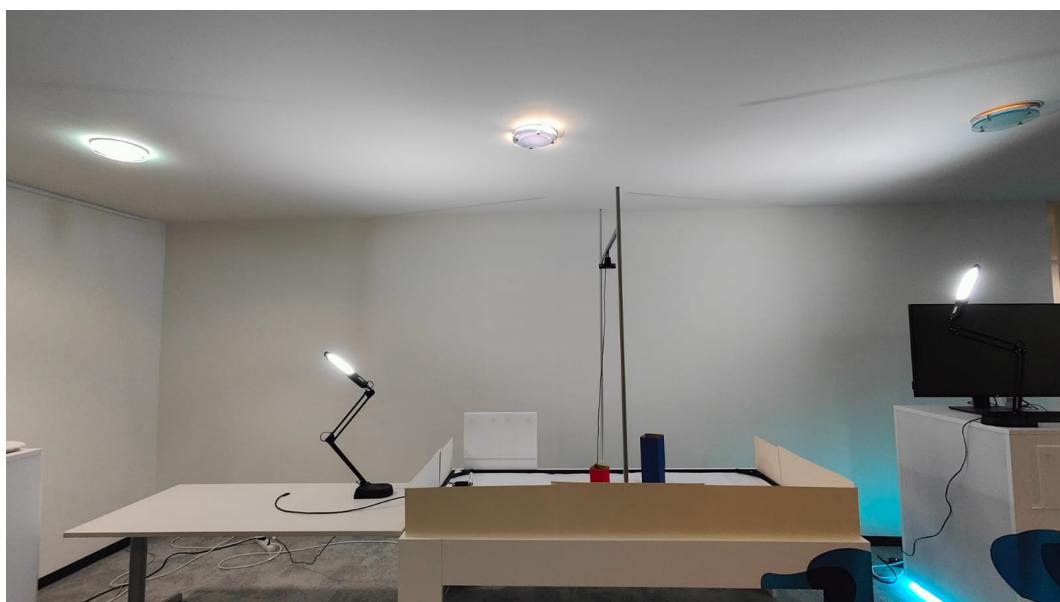


Abbildung 5.8.: Lichtsetup in der Realität

in unterschiedlicher Intensität auf die Arena, was zu einer unterschiedlich ausgeprägten Helligkeit auf den verschiedenen Seiten der Arena führt. Demzufolge werfen die aufgestellten Hindernisse und die angebrachte Pappmauer einen Schatten. Um diese Ungleichmäßigkeiten in der Helligkeit einzuschränken, wurden, wie man in Abbildung 5.8 sehen kann, zusätzliche Tischlampen aufgestellt. Dennoch muss ein größerer Bereich im HSV-Farbspektrum abgesucht werden, um die Hindernisse aus allen Perspektiven verlässlich erkennen zu können. Folglich ist es wichtig, Hindernisse mit stark abweichenden Farben zu wählen. Die Farbe Rot bietet sich auf den ersten Blick nicht an. Sie hat die besondere Eigenschaft im HSV-Farbraum sowohl einen geringen, als auch einen hohen Farbwert haben zu können (vgl. Abbildung 5.6). Um alle roten Farben aus dem Bild zu finden, muss man deshalb nach jedem Farbwert zwischen 0 und 179 suchen. Das führt jedoch dazu, dass beispielsweise auch helle oder dunkle Teile des blauen Hindernisses als rotes erkannt werden (siehe Abbildung 5.9). Dies kann umgangen werden, indem man nach Rot mit geringem und hohem Farbwert voneinander

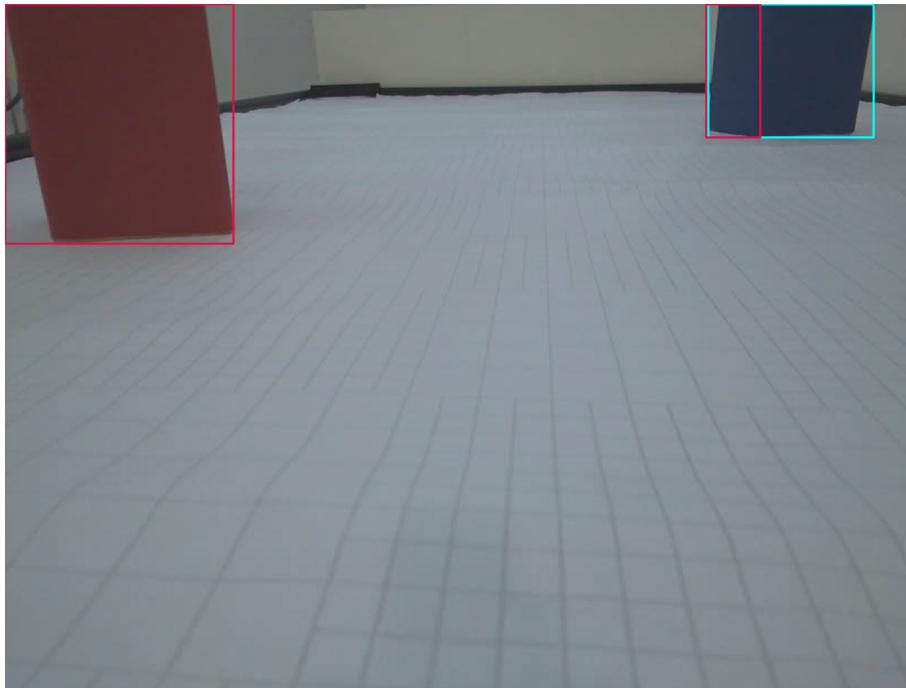
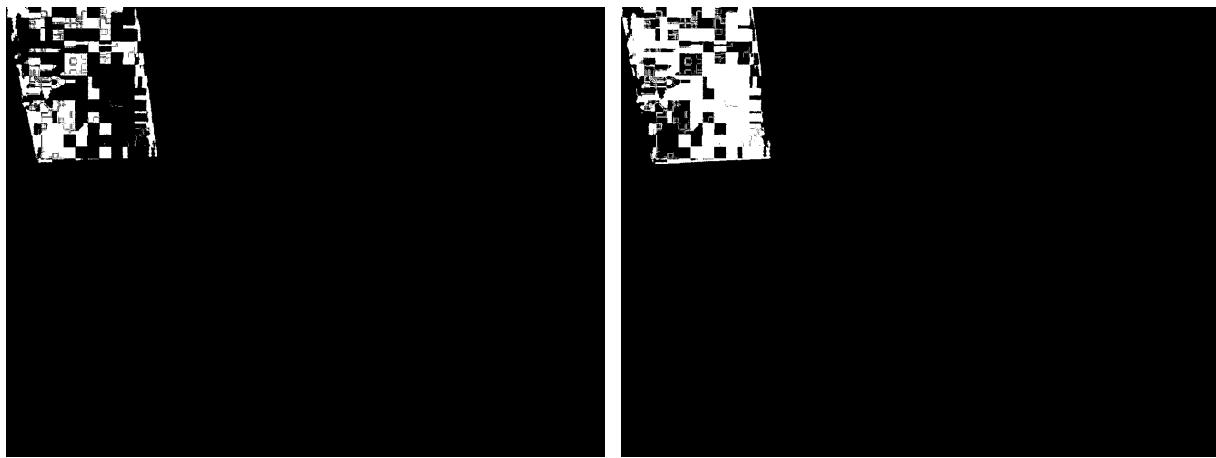


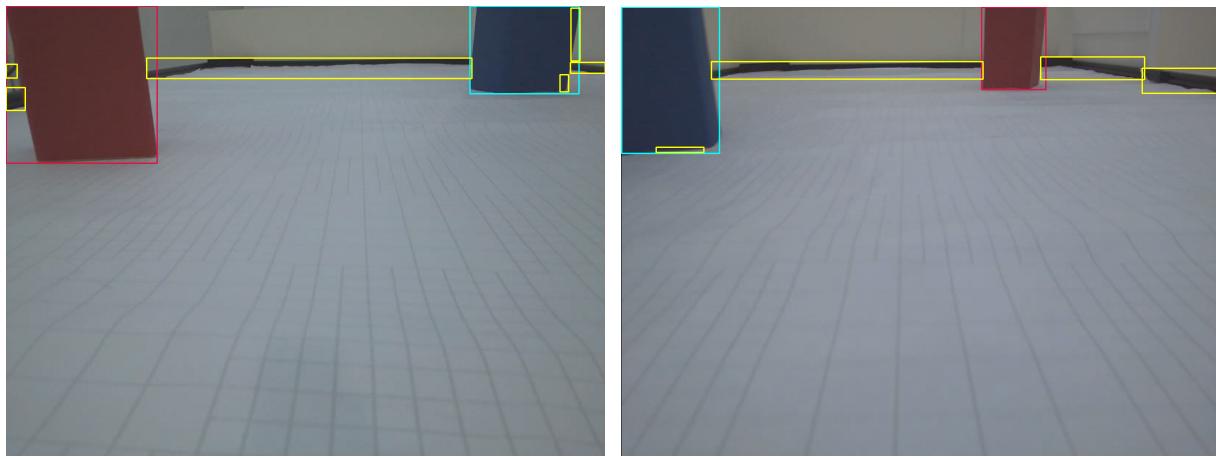
Abbildung 5.9.: Die Farbe rot wird auch im blauen Hindernis erkannt

getrennt sucht und die gefundenen Bildmasken zusammenfügt, bevor die Rechtecke der Hindernisse berechnet werden (siehe Abbildung 5.10). Weiterhin stellt die Farbe der Bande ein Problem dar. Sie nimmt, wie man in Abbildung 5.6 erkennen kann, einen großen Teil des HSV-Bereichs ein. Sie kann nahezu jeden Farbwert haben, solange die Sättigung und der Hellwert entsprechend gering sind. Ideal wäre es, eine andere Farbe zu nutzen. Dies ist nicht möglich, da die Farbe für eine andere Forschung, die an der Arena betrieben wird, wichtig ist. Stattdessen wird mit Hilfe der Tischlampen eine möglichst gleichmäßig helle Umgebung in der Arena geschaffen. Das Problem wird dadurch nicht vollständig beseitigt. Es werden auf den Bildern weiterhin einige dunkle Stellen fälschlich als Bande erkannt (siehe Abbildung 5.11).



(a) Bildmaske der Farbe rot mit hohem Farbwert      (b) Bildmaske der Farbe rot mit niedrigem Farbwert

Abbildung 5.10.: Bildmasken des roten Hindernisses mit hohem und niedrigem Farbwert



(a) Beispiel 1 (Rand des blauen Hindernisses)

(b) Beispiel 2 (Boden des blauen Hindernisses)

Abbildung 5.11.: Dunkle Stellen werden fälschlich als Bande wahrgenommen

## 5.3. Nutzung des neuronalen Netzes

Durch die Unterschiede zwischen der Simulation und der Realität, muss bei der Nutzung des Neuronalen Netzes einiges beachtet werden. Die Eingabedaten und die Ausgabedaten müssen jeweils angepasst werden, um das Netz sinnvoll einsetzen zu können.

### 5.3.1. Probleme beim neuronalen Netz

Das neuronale Netz, das in der Vorarbeit zu dieser Forschung entwickelt wurde, hat mehrere Fehler. So liefert jeder Ausgabewert eine negative Beschleunigung, unabhängig von den Eingabedaten (vgl. Abschnitt 5.3.2). Ob der berechnete Lenkwinkel sinnvoll ist, kann demzufolge nicht überprüft werden. Weiterhin wird eine Exception geworfen, wenn nur eine einzelne Bande und kein weiteres Hindernis erkannt wird. Eine solche Situation führt dazu, dass das Auto ohne zusätzliche Hilfe nicht weiter voran kommt. Demnach ist es nicht geeignet, mit dem Netz zu arbeiten. Aufgrund fehlender Reproduzierbarkeit der Simulation, war es im Rahmen dieser Forschung nicht möglich, ein neues

neuronales Netz zu trainieren. Stattdessen wurde mit dem synthetischen Daten gearbeitet, um in den folgenden Abschnitten eine weitere Herangehensweise vorschlagen zu können.

### 5.3.2. Unterschiede bei den Eingabedaten

Aufgrund mehrerer Unterschiede zwischen der Simulation und der Realität unterscheiden sich die Eingabedaten in den jeweiligen Setups. Während die Bilder in der Realität eine Auflösung von 816x616 Pixel haben, sind es in der Simulation nur 240x135 Pixel. Deshalb müssen die Werte der gefundenen Rechtecke auf die Auflösung der Bilder in der Trainingsumgebung des neuronalen Netzes skaliert werden, um von Diesem richtig verarbeitet werden zu können.

$$scaled\_x = \text{round}\left(\frac{240 \cdot x}{816}\right) \quad (5.6)$$

$$scaled\_y = \text{round}\left(\frac{135 \cdot y}{616}\right) \quad (5.7)$$

$$scaled\_w = \text{round}\left(\frac{240 \cdot w}{816}\right) \quad (5.8)$$

$$scaled\_h = \text{round}\left(\frac{135 \cdot h}{616}\right) \quad (5.9)$$

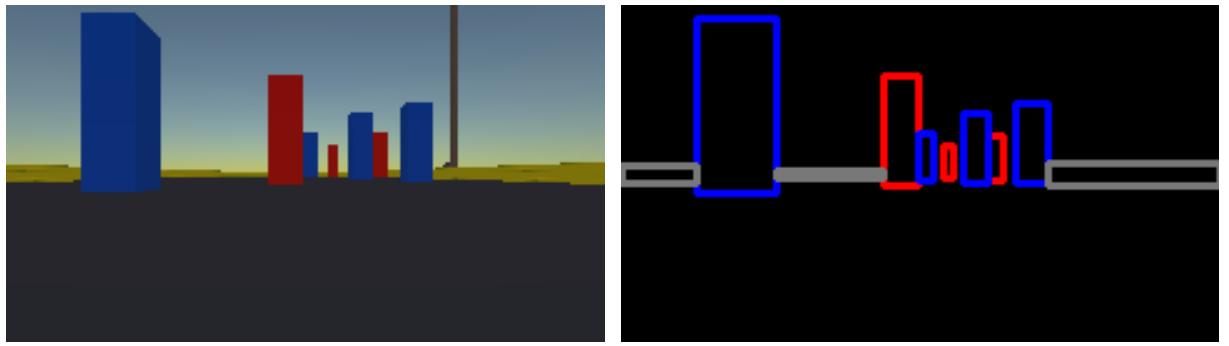
Weiterhin schaut die am Jetbot installierte Kamera, wie in Abschnitt 4.2, leicht nach unten. Zudem ist sie höher über dem Boden befestigt, als die Kamera in der Simulation. Aus diesem Grund werden die aufgestellten Objekte nicht vollständig auf den Bildern erkannt. Hinzu kommt, dass in der Realität Hindernisse genutzt wurden, die nicht die gleiche Form und Maße haben. Zusätzlich waren es quantitativ weniger als in der Simulation. Demzufolge ist es wichtig zu wissen, welchen Einfluss Abweichungen bei der Anzahl, Position, Höhe, oder Breite der Hindernisse auf die Berechnung der Ausgabedaten durch das Neuronale Netz haben. Dies wurde ermittelt, indem verschiedene Listen mit Tupeln der benötigten Form ( $x$ ,  $y$ ,  $w$ ,  $h$ ) erzeugt und an das Netz gegeben wurden. Diese Tupel stellen die durch die Computer Vision gefundenen Rechtecke der aufgestellten Hindernisse dar. Da es nicht auf die Richtigkeit der Ausgaben ankommt, konnte dafür das fehlerhafte Netz genutzt werden. Entscheidend ist zu erfahren, welchen Einfluss die genannten Unterschiede auf die Berechnung der Ausgabedaten haben können. Wie auch in der Simulation, wurde erst die Liste der roten, dann der blauen Hindernisse und zuletzt die Liste der Banden an das neuronale Netz übergeben. Zur Erzeugung der Testdaten wurde ein Bild, das in der Simulation aufgenommen wurde als Startpunkt definiert. Ein Beispiel für einen solchen Ausgangspunkt ist in Abbildung 5.12a dargestellt. Zu diesem Bild erhält man die in Listing 5.1 gezeigten Listen an Tupeln.

```

1  input1 = [[(129, 56, 4, 13), (147, 52, 6, 18), (105, 28, 14, 44)], [(119, 51, 6,
   19), (137, 43, 10, 28), (158, 39, 13, 32), (30, 5, 32, 70)], [(62, 66, 43, 3),
   (0, 64, 30, 7), (171, 63, 69, 9)]]
2
3  output1 = [-0.18434652071417346, -0.17600099359263346]
```

Listing 5.1: Eingabe und Ausgabe des Ausgangsbildes

Danach wurden händisch die Werte der gefundenen Hindernisse manipuliert, oder Elemente aus den Listen entfernt, um den Einfluss der Eingabedaten auf die Ausgabe zu evaluieren. Eine Erkenntnis ist, dass die Breite und die y-Koordinate eines Hindernisses keinen Einfluss auf die berechnete



(a) aufgenommenes Bild

(b) Veranschaulichung der Eingabedaten zu Listing 5.1

Abbildung 5.12.: Ausgangspunkt für Test der Eingabedaten

Ausgabe hat. So haben die Eingabedaten aus Listing 5.2 die gleiche Ausgabe, wie beim Ausgangspunkt.

```

1  input2 = [[(129, 0, 0, 13), (147, 0, 0, 18), (105, 0, 0, 44)], [(119, 0, 0, 19)
    , (137, 0, 0, 28), (158, 0, 0, 32), (30, 0, 0, 70)], [(62, 0, 0, 3), (0, 0, 0,
    7), (171, 0, 0, 9)]]

2

3  output2 = [-0.18434652071417346, -0.17600099359263346]
```

Listing 5.2: Eingabe und Ausgabe ohne Breite und y-Koordinate

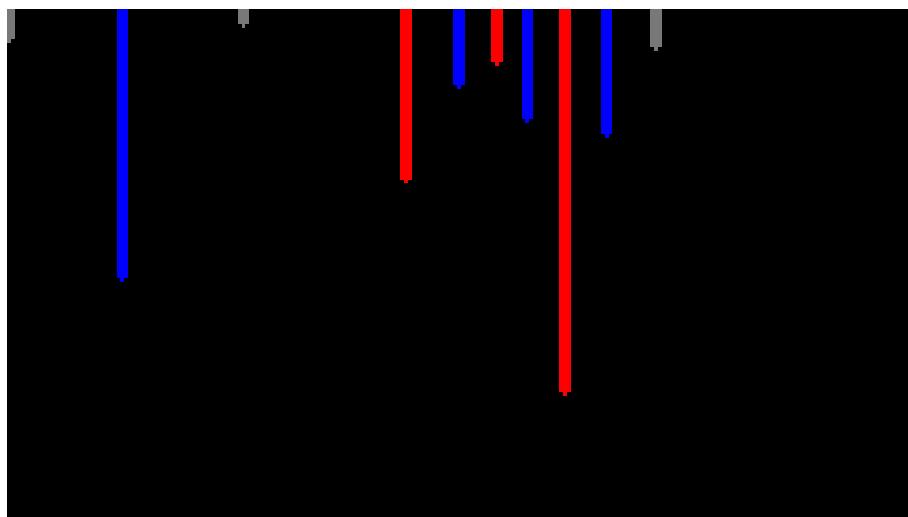


Abbildung 5.13.: Veranschaulichung der Eingabedaten in Listing 5.2

Das gleiche gilt für die gesamte Bande, unter der Bedingung, dass mindestens fünf rote und blaue Hindernisse erkannt werden (vgl. Listing 5.3).

```

1  input3 = [[(129, 56, 4, 13), (147, 52, 6, 18), (105, 28, 14, 44)], [(119, 51, 6,
    19), (137, 43, 10, 28), (158, 39, 13, 32), (30, 5, 32, 70)], []]

2

3  output3 = [-0.18434652071417346, -0.17600099359263346]
```

Listing 5.3: Eingabe und Ausgabe ohne Bande

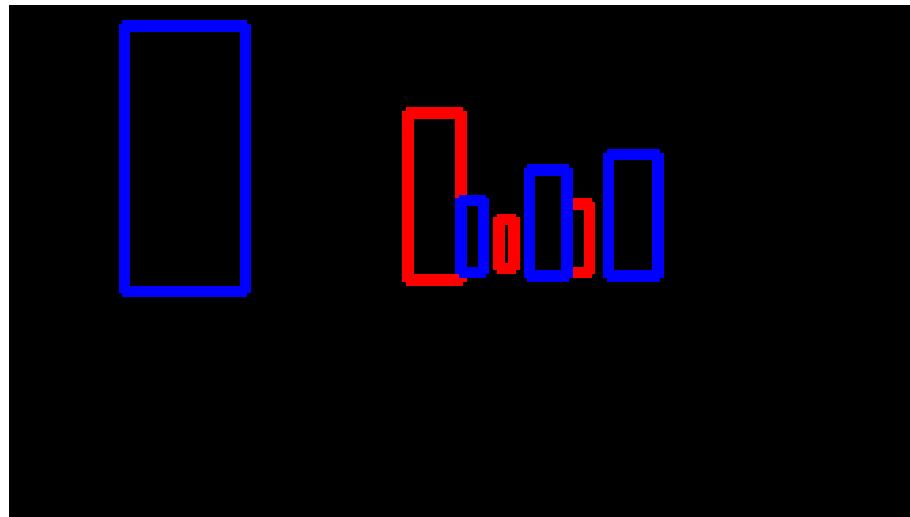


Abbildung 5.14.: Veranschaulichung der Eingabedaten in Listing 5.3

Wenn hingegen weniger als fünf rote und blaue Hindernisse erkannt werden und die Bande entfernt wird, soll das Auto weiter nach links lenken (vgl. Listings 5.4, 5.5). Die Beschleunigung bleibt beim Entfernen der Bande gleich.

```

1  input4 = [[(129, 56, 4, 13), (147, 52, 6, 18)], [(119, 51, 6, 19), (137, 43, 10,
    28)], [(62, 66, 43, 3), (0, 64, 30, 7), (171, 63, 69, 9)]]
2
3  output4 = [-0.11693620457548896, 0.012250203545667583]
```

Listing 5.4: Eingabe und Ausgabe bei begrenzter Anzahl an roten und blauen Hindernissen

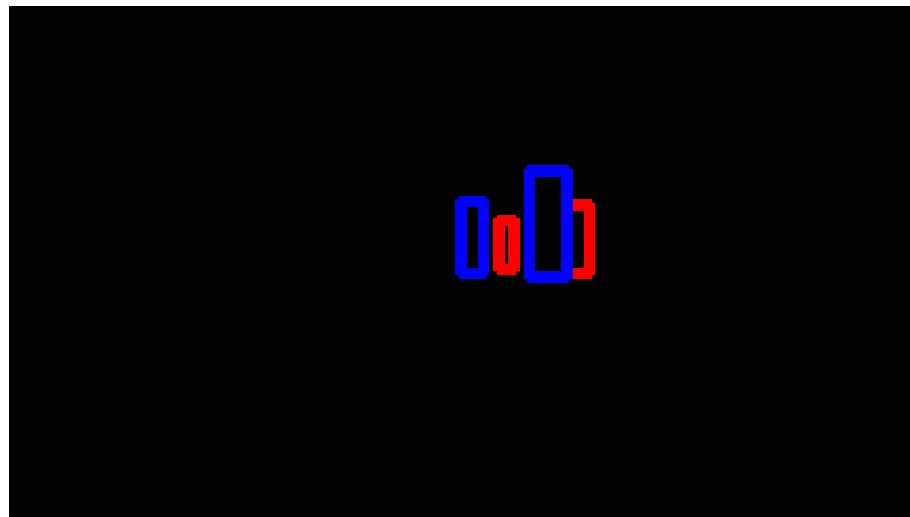


Abbildung 5.15.: Veranschaulichung der Eingabedaten in Listing 5.4

```

1  input5 = [[(129, 56, 4, 13), (147, 52, 6, 18)], [(119, 51, 6, 19), (137, 43, 10,
    28)], []]
2
3  output5 = [-0.11693620457548896, -0.1084201802214314]
```

Listing 5.5: Eingabe und Ausgabe bei begrenzter Anzahl an roten und blauen Hindernissen ohne Bande

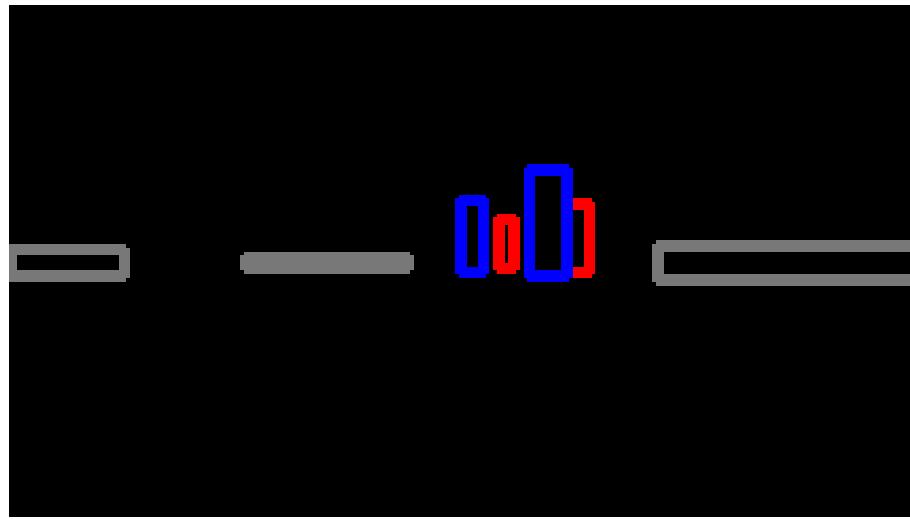


Abbildung 5.16.: Veranschaulichung der Eingabedaten in Listing 5.5

Allgemein spielt die Anzahl und das Verhältnis der erkannten roten und blauen Hindernisse eine entscheidende Rolle bei der Berechnung der Ausgabedaten. Wenn beispielsweise ein rotes oder ein blaues Hindernis entfernt wird, kann man große Veränderungen bei den berechneten Ausgaben für Beschleunigung und Lenkung feststellen. Entfernt man ein rotes Objekt aus der übergebenen Liste, so soll das Auto schneller abbremsen oder rückwärts fahren und stärker einlenken (siehe Listing 5.6). In Listing 5.7 erkennt man, dass es sich bei blauen Hindernissen umgekehrt verhält. Beim Entfernen eines blauen Objektes aus der Liste, wird eine geringere Abbremsung und Lenkung berechnet.

```

1  input6 = [[(129, 56, 4, 13), (147, 52, 6, 18)], [(119, 51, 6, 19), (137, 43, 10,
      28), (158, 39, 13, 32), (30, 5, 32, 70)], [(62, 66, 43, 3), (0, 64, 30, 7),
      (171, 63, 69, 9)]]

2

3  output6 = [-0.293150328659026, -0.28524620399450995]
```

Listing 5.6: Eingabe und Ausgabe, wenn ein rotes Hindernis entfernt wird

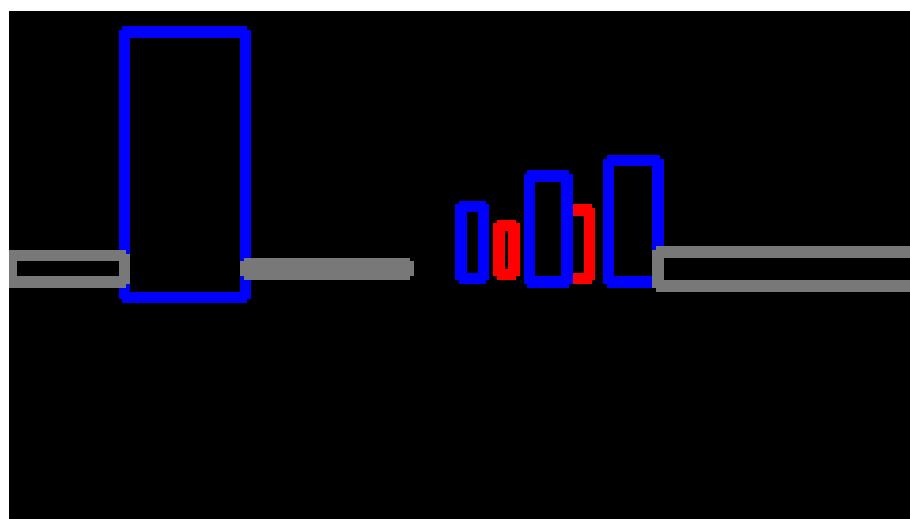


Abbildung 5.17.: Veranschaulichung der Eingabedaten in Listing 5.6

## 5. Simulation-to-Reality Gap

```

1  input7 = [[(129, 56, 4, 13), (147, 52, 6, 18), (105, 28, 14, 44)], [(119, 51, 6,
    19), (137, 43, 10, 28), (158, 39, 13, 32)], [(62, 66, 43, 3), (0, 64, 30, 7),
    (171, 63, 69, 9)]]
2
3  output7 = [-0.09593731405316916, -0.08738424512281204]
```

Listing 5.7: Eingabe und Ausgabe, wenn ein blaues Hindernis entfernt wird

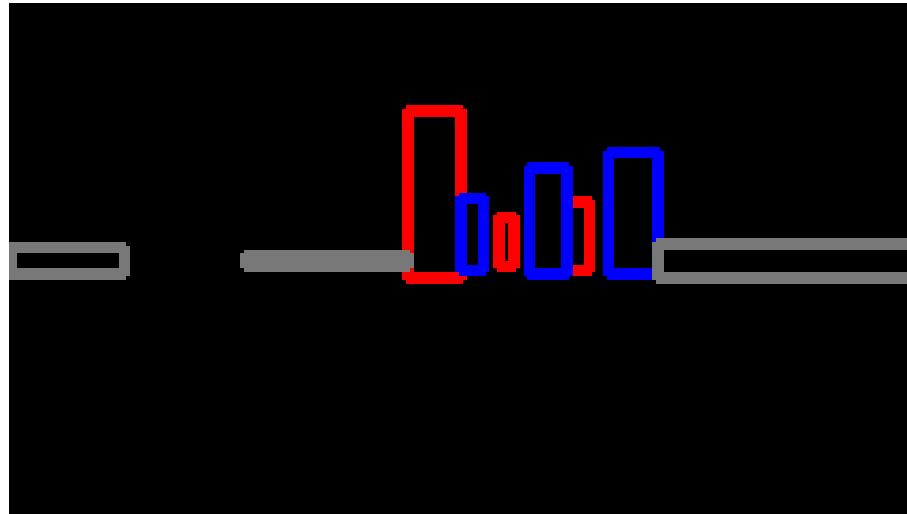


Abbildung 5.18.: Veranschaulichung der Eingabedaten in Listing 5.7

Demnach ist es ein Problem, dass in der Realität nur jeweils ein blaues und rotes Hindernis verwendet worden sind. Dadurch gibt es Einschränkungen zu möglichen Verhältnissen bei der Anzahl erkannter Objekte. Folglich können einige angelernte Verhaltensweisen des Fahrzeugs in der Simulation nicht in der realen Welt ausgenutzt werden. Weiterhin hat auch die erkannte Höhe des letzten roten und ersten blauen Hindernisses in den jeweiligen Listen einen Einfluss auf die berechneten Ausgabewerte. Diese entsprechen dem größten roten und kleinsten blauen Objekt in den Eingabedaten. Je höher das letzte rote Rechteck der Liste ist, desto weniger soll abgebremst und gelenkt werden (vgl. Listing 5.8).

```

1  input8 = [[(129, 56, 4, 13), (147, 52, 6, 18), (105, 28, 14, 100)], [(119, 51,
    6, 19), (137, 43, 10, 28), (158, 39, 13, 32), (30, 5, 32, 70)], [(62, 66, 43,
    3), (0, 64, 30, 7), (171, 63, 69, 9)]]
2
3  output8 = [-0.14826278343196347, -0.13981620455128435]
```

Listing 5.8: Eingabe und Ausgabe bei höherem letzten roten Hindernis

Die Höhe des ersten blauen Rechtecks in der Eingabe hat einen geringeren Einfluss. Dennoch wird eine stärkere Bremsung und Lenkung berechnet, je höher das entsprechende Objekt wahrgenommen wird (vgl. Listing 5.9).

```

1  input9 = [[(129, 56, 4, 13), (147, 52, 6, 18), (105, 28, 14, 44)], [(119, 51, 6,
    80), (137, 43, 10, 28), (158, 39, 13, 32), (30, 5, 32, 70)], [(62, 66, 43, 3),
    (0, 64, 30, 7), (171, 63, 69, 9)]]
2
```

## 5. Simulation-to-Reality Gap

```
3     output9 = [-0.18982061868154185, -0.18149239258187022]
```

Listing 5.9: Eingabe und Ausgabe bei höherem ersten blauen Hindernis

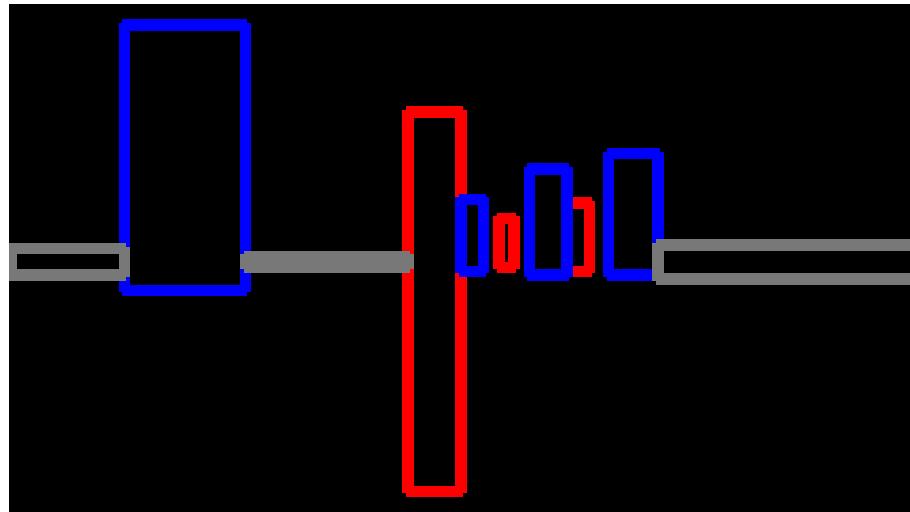


Abbildung 5.19.: Veranschaulichung der Eingabedaten in Listing 5.8

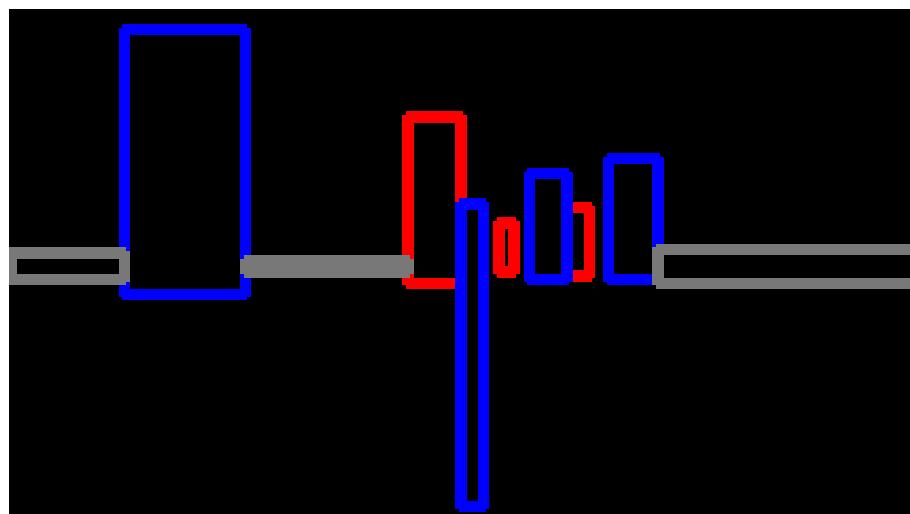


Abbildung 5.20.: Veranschaulichung der Eingabedaten in Listing 5.9

Die Höhen der anderen erkannten Rechtecke spielen keine Rolle (siehe Listing 5.10).

```
1   input10 = [[(129, 56, 4, 1), (147, 52, 6, 1), (105, 28, 14, 44)], [(119, 51, 6, 19), (137, 43, 10, 1), (158, 39, 13, 1), (30, 5, 32, 1)], [(62, 66, 43, 1), (0, 64, 30, 1), (171, 63, 69, 1)]]  
2  
3   output10 = [-0.18434652071417346, -0.17600099359263346]
```

Listing 5.10: Eingabe und Ausgabe mit manipulierten Höhen

Da die Höhe der Hindernisse dennoch einen Einfluss auf die Berechnung der Ausgabedaten hat, sollte die Kamera, wie in der Simulation, knapp über und parallel zur Oberfläche der Arena, an das Fahrzeug angebracht werden. Dadurch können die Hindernisse vollständig erkannt werden. Die genaue Höhe, in welcher die Kamera angebracht werden müsste, ist nicht aus der Simulation zu

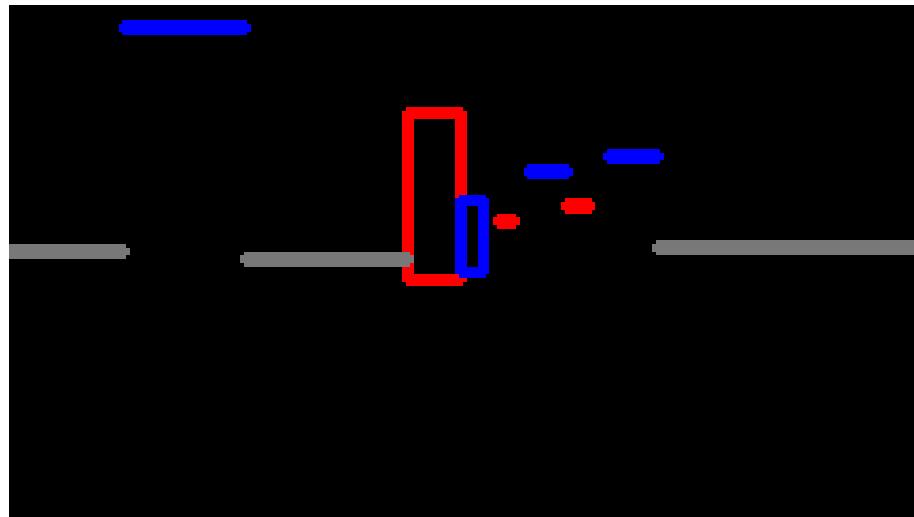


Abbildung 5.21.: Veranschaulichung der Eingabedaten in Listing 5.10

entnehmen (siehe Abschnitt 4.2.1). Dies war in dieser Forschung nicht möglich, da die Mauer, die den Hintergrund verdeckt, zu klein ist. Somit werden bei der beschriebenen Positionierung der Kamera erneut Objekte außerhalb der Arena als Hindernisse erkannt. (siehe Abbildung 5.22). Ei-

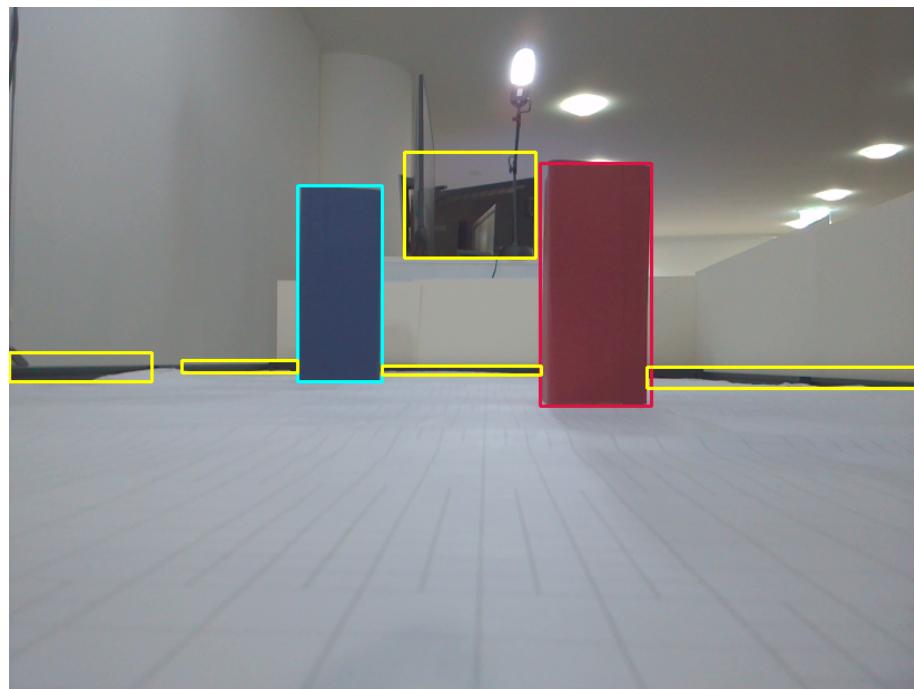


Abbildung 5.22.: Objekte im Hintergrund sind wieder erkennbar, wenn man die Position der Kamera an die Simulation anpasst

ne höhere Trennmauer konnte nicht ohne größeren Aufwand organisiert werden. Außerdem hätte sie die Lichtquellen blockiert und folglich die Arena verdunkelt. Das hätte wiederum die Hinderniserkennung erschwert, da dunklere Farben, schwerer zu einem bestimmten Farbwert zugeordnet werden können. Weiterhin ist in Abbildung 5.22 erkennbar, dass bei dieser Positionierung der Kamera Unebenheiten auf der Oberfläche die Bände an einigen Stellen verdecken. Dadurch wird die Bände nicht mehr richtig von der Computer Vision erkannt.

### 5.3.3. Verarbeitung der Outputs

#### 5.3.3.1. Beschleunigung

Der erste Ausgabewert ist die Beschleunigung des Fahrzeugs. Der JetBot besitzt jedoch keine Beschleunigung im herkömmlichen Sinne. Der Motortreiber kann lediglich einen konstanten Wert in ein Signal für die Gleichstrommotoren umwandeln, die die Räder zum Drehen bringen. Somit muss mithilfe des ausgegebenen Wertes und der Geschwindigkeit, mit der sich der JetBot im Moment fortbewegt, eine neue Geschwindigkeit berechnet werden. Die zurückgegebene Beschleunigung des neuronalen Netzes ist ein abstrakter Wert zwischen -1 und 1. Der Controller kann eine natürliche Zahl zwischen 0 und 255 als Wert erhalten und jeweils entscheiden, ob sich die Räder mit dem gesetzten Wert nach vorne, oder nach hinten drehen sollen. Die Ausgabe des Netzes wird zur Nutzung auf dieses Intervall skaliert, indem sie mit 255 multipliziert wird.

$$scaled\_output = output \cdot 255 \quad (5.10)$$

Die Berechnung für die Geschwindigkeit des JetBot sieht dann folgendermaßen aus:

$$new\_speed = current\_speed + scaled\_output \cdot time\_since\_last\_output \quad (5.11)$$

Sollte das Ergebnis positiv sein, fährt der JetBot vorwärts, und wenn es negativ ist, fährt er rückwärts. Dabei gibt es mehrere Probleme, die nur in der Realität auftreten. In der Simulation wird keine Reibung berücksichtigt. Bevor sich das Auto in der realen Welt fortbewegen kann, müssen verschiedene Arten der Reibung, vor allem die Haftriebung, überwunden werden. Dafür muss ein gewisser Mindeststrom an die Motoren übertragen werden. Je nach Oberfläche, auf der das Fahrzeug fährt, muss dieser unterschiedlich gewählt werden. In dieser Forschung variierte der kleinstmögliche Wert, der dem Motor Controller übergeben wurde, zwischen 60 und 80. Bei einem kleineren Wert stand der JetBot still. Die Varianz begründet sich durch Unebenheiten auf der Oberfläche. Wie man in Abbildung 5.23 erkennen kann, befinden sich kleine Wellen auf der Plane, mit der die Arena überdeckt ist. Über diese Wellen muss der JetBot gelegentlich fahren, um voranzukommen. Zur Überwindung dieser Stellen wird mehr Kraft benötigt. Folglich konnten Situationen, in denen das Auto in der Simulation nur sehr langsam fährt, nicht gleichermaßen in die Realität übertragen werden.

#### 5.3.3.2. Lenkung

Der größte Unterschied zwischen der Simulation und der Realität in dieser Forschung, ist die Lenkweise der Fahrzeuge. In der Simulation wird eine herkömmliche Lenkung nachgeahmt, bei der die Vorderräder um einen bestimmten Winkel eingeschlagen werden. Bei dem JetBot funktioniert das Lenken anders. Die Räder des JetBot können nicht nach außen oder innen rotiert werden. Da die beiden Räder von zwei jeweils unabhängigen Motoren gedreht werden, wird stattdessen gelenkt, indem sich ein Rad schneller dreht, als das andere. Wenn das linke Rad schneller dreht, lenkt das Auto nach rechts. Wenn hingegen das rechte Rad schneller rotiert, wird nach links gelenkt. Der

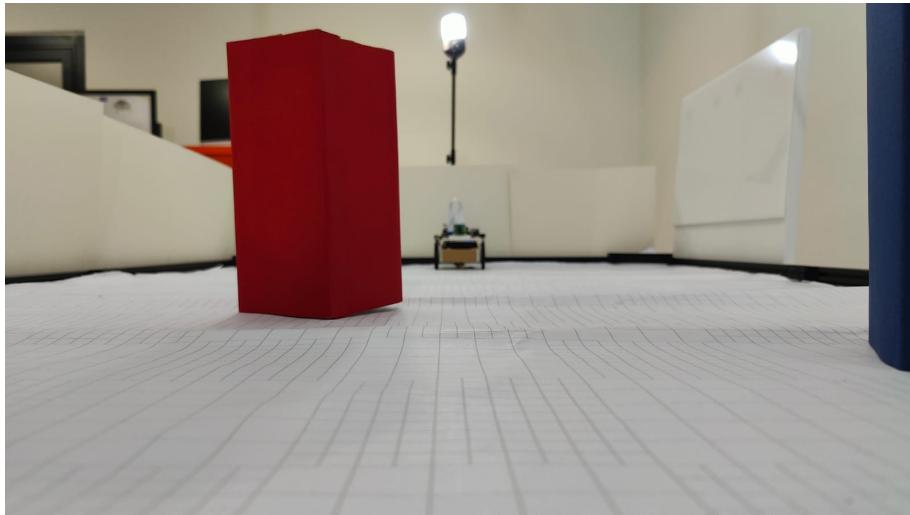


Abbildung 5.23.: Wellen auf der Oberfläche der Arena

berechnete Wert für den Lenkwinkel muss somit erst in sinnvolle Werte für die Motoren umgewandelt werden. Wenn nur ein Rad dreht, bewegt es sich kreisförmig um das andere. Bei einer festen Winkelgeschwindigkeit  $\omega$  kann man die benötigte Zeit  $t$  für eine Drehung um einen gewünschten Winkel  $\phi$  berechnen. Die Winkelgeschwindigkeit wurde ermittelt, indem händisch die Zeit gestoppt wurde, die der JetBot benötigt um sich einmal vollständig im Kreis zu drehen. Bei verschiedenen Geschwindigkeitswerten konnten die in Tabelle 5.1 dargestellten Zeiten und Winkelgeschwindigkeiten ermittelt werden. Die Winkelgeschwindigkeiten wurden wie folgt berechnet:

$$\omega = \frac{2 \cdot \pi}{T} \quad (5.12)$$

Übergebener Wert	Zeit $T$ für eine Umdrehung mit linkem Rad	Zeit $T$ für eine Umdrehung mit rechtem Rad	Winkelgeschw. $\omega$ linkes Rad	Winkelgeschw. $\omega$ rechtes Rad
$0.25 \cdot 255 = 63$	6.76s	7.06s	$0.93\frac{1}{s}$	$0.89\frac{1}{s}$
$0.3 \cdot 255 = 76$	4.9s	5.4s	$1.28\frac{1}{s}$	$1.16\frac{1}{s}$
$0.5 \cdot 255 = 127$	2.46s	2.46s	$2.55\frac{1}{s}$	$2.55\frac{1}{s}$
$0.75 \cdot 255 = 191$	1.54s	1.54s	$4.08\frac{1}{s}$	$4.08\frac{1}{s}$
$1 \cdot 255 = 255$	1.23s	1.23s	$5.11\frac{1}{s}$	$5.11\frac{1}{s}$

Tabelle 5.1.: gemessene Winkelgeschwindigkeiten bei verschiedenen übergebenen Werten

Beim Drehen nach vorne oder hinten gibt es dabei keine Unterschiede. Die benötigte Zeit, um einen bestimmten Winkel  $\phi$  zu drehen, kann mit Formel 5.13 berechnet werden. Dafür benötigt man den Winkel in Bogenmaß  $\phi_{rad} = \frac{\phi \cdot \pi}{180}$ .

$$t = \frac{\phi_{rad}}{\omega} \quad (5.13)$$

Auffallend ist, dass die Geschwindigkeiten der jeweiligen Räder bei geringeren Geschwindigkeitswerten unterschiedlich sind. Um die beiden Räder aneinander anzupassen, müssen weitere Winkelgeschwindigkeiten ermittelt werden. Dann können für jedes Rad jeweils Werte mit gleicher Ge-

schwindigkeit gewählt werden. Andernfalls driftet der JetBot nach rechts, wenn beiden Motoren der gleiche geringe Wert übergeben wird. Weitherhin wächst die benötigte Zeit für eine ganze Umdrehung mit steigendem Wert nicht linear. Es bietet sich an, die ermittelten Werte zu nutzen und aus dem Stand den Winkel einzulenken. Anschließend können einige Zentimeter geradeaus gefahren werden, bis der nächste ausgegebene Winkel gelenkt wird. Um diese Herangehensweise zu testen, wurden die benötigte Drehzeiten, wie beschrieben, für verschiedene Lenkwinkel ermittelt. Zum Lenken wurde die höchstmögliche Geschwindigkeit des drehenden Rades gewählt, was dem Wert 255 entspricht. Die dazugehörige Winkelgeschwindigkeit beträgt  $5.11\frac{1}{s}$ . Aus einem festen Startpunkt heraus, sollte das Auto jeweils  $90^\circ$ ,  $180^\circ$  und  $360^\circ$  nach rechts und links lenken. Um die gefahrenen Winkel messen zu können, hat sich das Fahrzeug auf einem Papier gedreht. Dadurch konnten Ausgangs- und Endpunkt der Drehung markiert werden. Die durchschnittlich gemessenen Werte sind in Abbildung 5.24 dargestellt. Die Punkte A und B stellen dabei die Fixpunkte bei den

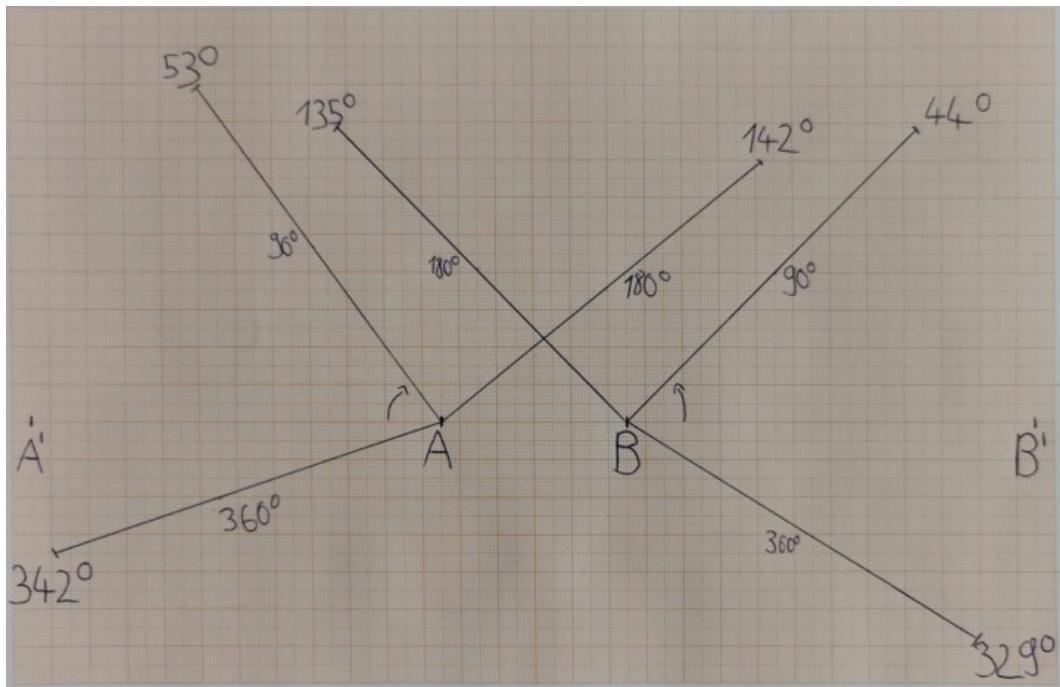


Abbildung 5.24.: Gemessene Winkel beim Testen der Lenkung

Lenkungen nach rechts (A) und links (B) dar. Diese Punkte entsprechen den Positionen des sich nicht rotierenden Rades. A' und B' sind die Startpunkte des sich drehenden Rades bei der Lenkung. Die eingezeichneten Linien führen zu den erreichten Endpunkten des Rades bei einem gewünschten Winkel von jeweils  $90^\circ$ ,  $180^\circ$  und  $360^\circ$ . Die erreichten Punkte sind mit den tatsächlich gedrehten Winkeln beschrieben. Bei weiteren Wiederholungen der Tests konnte eine Abweichung von bis zu  $6^\circ$  bei den gefahrenen Winkeln festgestellt werden. Allgemein kann man erkennen, dass stets zu wenig gelenkt wird. Bei einer Lenkung nach links fehlen bis zu  $46^\circ$ , um den gewünschten Winkel zu erreichen. Bei der Lenkung nach rechts sind es mit bis zu  $37^\circ$  weniger. Selbst eine Lenkung um  $360^\circ$  kann nicht vollständig erzielt werden, obwohl die Winkelgeschwindigkeiten aus dieser Drehung ermittelt wurden. Beim Lenken nach rechts wurden dabei  $342^\circ$ , und nach links  $329^\circ$  erreicht. Somit sind die Abweichungen zu den tatsächlich erreichten Winkeln bei einer gewünschten Lenkung um  $360^\circ$  am geringsten. Zu den Unterschieden zwischen den gewünschten und gefahrereren Lenkwinkeln, kommt es zum einen dadurch, dass die Zeiten für eine ganze Umdrehung mit der Hand gemessen

## *5. Simulation-to-Reality Gap*

---

wurden. Demnach sind die gemessenen Werte ungenau. Diese Ungenauigkeit wird durch Rundungen weiter verstrkt. Zustzlich spielt auch die Laufzeit des verwendeten Skriptes eine Rolle. Die genaue Zeit, die fr die Dauer der Drehung gegeben ist, wird nicht richtig eingehalten. Folglich ist die Genauigkeit bei kleineren Lenkgeschwindigkeiten auf Kosten der Drehgeschwindigkeit hher. Dies begrndet ebenso die Varianz der erreichten Winkel bei mehreren Versuchen des Tests.

## 6. Evaluierung

### 6.1. Ergebnisse der Computer Vision

Bei der Bildverarbeitung konnten gute Ergebnisse erzielt werden. Das Rauschen auf den aufgenommenen Bildern wurde vollständig beseitigt. Weiterhin konnten, nachdem die Hinderniserkennung angepasst und der Hintergrund verdeckt wurde, das rote und blaue Objekt zuverlässig richtig erkannt werden. Lediglich die Farbe der Bande stellte bei der Hinderniserkennung ein Problem dar. Sie nimmt ein zu großes Spektrum im HSV-Bereich ein. Somit konnten die HSV-Schwellenwerte für die Suche der Bande im Bild nicht ausreichend genau gewählt werden. Folglich wurden einige Stellen im Bild fälschlich als solche erkannt (siehe Abbildung 5.11). Außerdem wurde die Bande nicht immer vollständig erkannt, da sie von Unebenheiten in der Oberfläche leicht verdeckt wurde. Das führte dazu, dass sie an diesen Stellen als sehr klein wahrgenommen wurde. Da zur Beseitigung des Rauschens eine Mindestfläche für die erkannten Rechtecke gesetzt wurde, wurden diese Teile der Bande folglich ausgeschlossen. So kommt es dazu, dass statt einer zusammenhängenden Bande, zwei oder mehr erkannt wurden (siehe Abbildung 5.11b, 5.22).

### 6.2. Ergebnisse bei der Übertragung des Neuronalen Netzes

Bei dem Setup in der Realität gibt es viele kritische Unterschiede zur Simulation, die die Nutzung des Neuronalen Netzes beeinträchtigt haben. Durch die Computer Vision konnten die nötigen Ausgabedaten aus den Bildern extrahiert werden, jedoch unterscheiden sich diese aufgrund der Anzahl der aufgestellten Objekte und der Positionierung der Kamera von denen in der Simulation. Die Methoden, die dieses Problem beheben konnten, haben wiederum für neue Komplikationen gesorgt. Im Endeffekt konnte deshalb keine Lösung dafür gefunden werden. Das führte dazu, dass bei der Berechnung der Ausgabedaten einige erlernte Muster bezüglich der Höhe und Anzahl der erkannten Hindernisse nicht richtig ausgenutzt werden konnten. Die Verarbeitung der Ausgabedaten des Neuronalen Netzes stellte den größten Teil der Reality Gap in dieser Forschung dar. Hauptgrund dafür sind die Unterschiede der Fahrzeuge in Simulation und Realität. Die zurückgegebene Beschleunigung konnte in einen vom Motortreiber nutzbaren Wert umgewandelt werden. Es ist jedoch unklar, ob sich das Fahrzeug in der Realität bei identischen Geschwindigkeitswerten ebenso schnell fortbewegt, wie in der Simulation. Das liegt daran, dass in beiden Umgebungen der Wert für die Geschwindigkeit abstrakt ist. Zusätzlich wurde in der Simulation keine Reibung berücksichtigt. In der Realität muss das Fahrzeug erst die Haftreibung überwinden, bevor es anfängt sich fortzubewegen. Dadurch konnten Situationen, in denen der JetBot sehr langsam fahren soll, nicht in die Realität übertragen werden. Weiterhin ist die für die Lenkung vorgeschlagene Methode fehlerhaft und bedarf Verbesserung. Die gewünschten Lenkwinkel konnten in der Praxis nicht erreicht werden. Außerdem wurden die Tests für die Lenkung unter der Annahme durchgeführt, dass ein Rad stillsteht und somit aus dem Stand gelenkt wird. Beim autonomen Fahren sollte das Fahrzeug nicht stehen bleiben, um zu Lenken. Die Fortbewegung ist sonst stockend und nicht effizient. Die vorgeschlagene Lenkmethode eignet sich jedoch nicht für eine Lenkung während der Fortbewegung.

## *6. Evaluierung*

---

Sie würde zu viel Platz benötigen und zu viel Zeit in Anspruch nehmen. Je schneller das Auto fährt, desto größer ist der Platzaufwand und die Lenkdauer. Dies liegt daran, dass das lenkende Rad aufgrund der Maximalgeschwindigkeit der Motoren nur begrenzt schneller rotieren kann, als das andere. Hinzu kommt, dass mit synthetischen Daten und nicht mit den tatsächlichen Ausgaben des Neuronalen Netzes gearbeitet wurde. Es wird für jedes aufgenommene Bild eine Ausgabe berechnet. Bei einer Bildwiederholfrequenz von 30 Bildern pro Sekunde müssen demnach jede Sekunde 30 Ausgaben auf die Räder übertragen werden, um die Ergebnisse der Simulation zu erzielen. Das Abschließen einer Lenkung dauert jedoch länger als die Berechnung neuer Ausgabedaten. Das führt dazu, dass das Auto ununterbrochen auf der Stelle dreht, ohne voranzukommen, wenn alle neu berechneten Werte verarbeitet werden sollen.

## 7. Zusammenfassung und Ausblick

### 7.1. Zusammenfassung

In dieser Arbeit wurde ein in einer Simulation entwickeltes Neuronales Netz auf einen echten, physischen Roboter übertragen. Dabei handelt es sich um ein Fahrzeug, das mit einer Kamera ausgestattet ist. Es sollte schnellstmöglich von einer Seite einer Arena zur anderen fahren und dabei aufgestellte Hindernisse erkennen und abhängig von deren Farbe umfahren. Die größten Probleme der Hinderniserkennung konnten dabei bewältigt werden. Letztendlich ist es jedoch nicht gelungen, das gewünschte Verhalten des Fahrzeugs in der Realität zu erzielen. Grund dafür ist vor allem, dass das gegebene Neuronale Netz aufgrund mehrerer Fehler ungeeignet ist. Weiterhin gibt es starke Unterschiede bei der Lenkweise zwischen dem Fahrzeug in der Simulation und in der Realität. Dadurch konnte keine gute Lösung gefunden werden, die ausgegebenen Lenkwinkel in der echten Welt umzusetzen.

### 7.2. Ausblick

#### 7.2.1. Computer Vision

Um ein besseres Ergebnis bei der Hinderniserkennung zu erzielen, sollte man sich nicht auf die Farbsuche im Bild verlassen. Das Szenario innerhalb der Arena ist gegenüber echter Fahrszenarien stark vereinfacht. Dennoch kam es in dieser Forschung zu Problemen bei der Erkennung der aufgestellten Objekte. Folglich ist es nicht geeignet, diese Methode der Objekterkennung in weiteren Forschungen zum autonomen Fahren zu nutzen. Statt nach Farben zu suchen, könnte eine künstliche Intelligenz trainiert werden, die Objekte automatisch erkennen kann. Da jedoch nicht nach Mustern oder Zusammenhängen in Bildern gesucht wird, sondern nach konkreten Objekten, kann das dafür notwendige maschinelle Lernen nicht unüberwacht erfolgen. Folglich müssen unzählige Daten händisch beschriftet werden, um Datensätze zum Trainieren der künstlichen Intelligenz zu erstellen. Das ist mit viel zeitlichem Aufwand verbunden.

#### 7.2.2. Anpassung der Simulation und des Neuronalen Netzes

Allgemein sollte nicht die Realität nach der Simulation entwickelt werden. Es ist effizienter, die Simulation an die Realität anzupassen. Man könnte die Funktionsweise des JetBot simulieren und bei gleichen Eingabedaten Ausgabewerte für die jeweiligen Geschwindigkeiten der beiden Räder berechnen. Außerdem ist es sinnvoll, die Mindestgeschwindigkeit, die aufgrund der Reibung benötigt wird, in der Simulation zu übernehmen. Dadurch erspart man sich Arbeit bei der Übertragung des trainierten Neuronalen Netzes in die Realität, da die ausgegebenen Werte direkt an die Motoren weitergegeben werden können. Man müsste sich folglich insbesondere nicht mit der Übertragung der Beschleunigung und Lenkweise auf das reale Fahrzeug auseinandersetzen, da ergänzende Berechnungen zu beiden Werten wegfallen. Weiterhin sollte auf die Laufzeit des Neuronalen Netzes

## *7. Zusammenfassung und Ausblick*

---

geachtet werden. Im Fall, dass zu frequent neue Werte für die Radgeschwindigkeiten übermittelt werden, könnte es zu ungewollten Verhalten der Räder kommen.

## Literatur

- [1] Afsoon Afzal u. a. „A Study on the Challenges of Using Robotics Simulators for Testing“. In: *CoRR* abs/2004.07368 (2020). arXiv: 2004 . 07368. URL: <https://arxiv.org/abs/2004.07368>.
- [2] Anders Krogh. „What are artificial neural networks?“ In: *Nature Biotechnology* 26.2 (2008), S. 195–197. ISSN: 1546-1696. DOI: 10 . 1038/nbt1386. URL: <https://doi.org/10.1038/nbt1386>.
- [3] In: () .
- [4] Seyed Jalaleddin Mousavirad u. a. „An Effective Hybrid Approach for Optimising the Learning Process of Multi-layer Neural Networks“. In: *Advances in Neural Networks – ISNN 2019*. Hrsg. von Huchuan Lu, Huajin Tang und Zhanshan Wang. Cham: Springer International Publishing, 2019, S. 309–317. ISBN: 978-3-030-22796-8.
- [5] Josh C. Bongard. „Evolutionary Robotics“. In: *Commun. ACM* 56.8 (2013), 74–83. ISSN: 0001-0782. DOI: 10.1145/2493883. URL: <https://doi.org/10.1145/2493883>.
- [6] Deborah Walters. „Computer Vision“. In: *Encyclopedia of Computer Science*. GBR: John Wiley und Sons Ltd., 2003, 431–435. ISBN: 0470864125.
- [7] Fahad Shahbaz Khan u. a. „Color attributes for object detection“. In: *2012 IEEE Conference on Computer Vision and Pattern Recognition*. 2012, S. 3306–3313. DOI: 10.1109/CVPR.2012.6248068.
- [8] Kayla Cameron und Md Shafiul Islam. „Multiple Objects Detection using HSV“. In: *NAECON 2018 - IEEE National Aerospace and Electronics Conference*. 2018, S. 270–273. DOI: 10.1109/NAECON.2018.8556711.
- [9] Boguslaw Cyganek. *Object detection and recognition in digital images: theory and practice*. John Wiley & Sons, 2013.
- [10] Nick Jakobi, Phil Husbands und Inman Harvey. „Noise and the reality gap: The use of simulation in evolutionary robotics“. In: *Advances in Artificial Life*. Hrsg. von Federico Morán u. a. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, S. 704–720. ISBN: 978-3-540-49286-3.
- [11] Nick Jakobi. „Running across the reality gap: Octopod locomotion evolved in a minimal simulation“. In: *Evolutionary Robotics*. Hrsg. von Philip Husbands und Jean-Arcady Meyer. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, S. 39–58. ISBN: 978-3-540-49902-2.
- [12] Joonho Lee u. a. „Learning quadrupedal locomotion over challenging terrain“. In: *Science Robotics* 5.47 (2020), eabc5986. DOI: 10.1126/scirobotics.abc5986. eprint: <https://www.science.org/doi/pdf/10.1126/scirobotics.abc5986>. URL: <https://www.science.org/doi/abs/10.1126/scirobotics.abc5986>.
- [13] Erica Salvato u. a. „Crossing the Reality Gap: A Survey on Sim-to-Real Transferability of Robot Controllers in Reinforcement Learning“. In: *IEEE Access* 9 (2021), S. 153171–153187. DOI: 10.1109/ACCESS.2021.3126658.

- [14] Wenshuai Zhao, Jorge Peña Queralta und Tomi Westerlund. „Sim-to-Real Transfer in Deep Reinforcement Learning for Robotics: a Survey“. In: *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*. 2020, S. 737–744. DOI: [10.1109/SSCI47803.2020.9308468](https://doi.org/10.1109/SSCI47803.2020.9308468).
- [15] Jack Collins, David Howard und Jurgen Leitner. „Quantifying the Reality Gap in Robotic Manipulation Tasks“. In: *2019 International Conference on Robotics and Automation (ICRA)*. 2019, S. 6706–6712. DOI: [10.1109/ICRA.2019.8793591](https://doi.org/10.1109/ICRA.2019.8793591).
- [16] Sylvain Koos, Jean-Baptiste Mouret und Stéphane Doncieux. „The Transferability Approach: Crossing the Reality Gap in Evolutionary Robotics“. In: *IEEE Transactions on Evolutionary Computation* 17.1 (2013), S. 122–145. DOI: [10.1109/TEVC.2012.2185849](https://doi.org/10.1109/TEVC.2012.2185849).
- [17] Juan Cristóbal Zagal, Javier Ruiz del Solar und Paul Vallejos. „Back to reality: Crossing the reality gap in evolutionary robotics“. In: *IFAC Proceedings Volumes* 37.8 (2004). IFAC/EU-RON Symposium on Intelligent Autonomous Vehicles, Lisbon, Portugal, 5-7 July 2004, S. 834–839. ISSN: 1474-6670. DOI: [https://doi.org/10.1016/S1474-6670\(17\)32084-0](https://doi.org/10.1016/S1474-6670(17)32084-0). URL: <https://www.sciencedirect.com/science/article/pii/S1474667017320840>.
- [18] Antoine Ligot und Mauro Birattari. „Simulation-only experiments to mimic the effects of the reality gap in the automatic design of robot swarms“. In: *Swarm Intelligence* 14.1 (2020), S. 1–24. ISSN: 1935-3820. DOI: [10.1007/s11721-019-00175-w](https://doi.org/10.1007/s11721-019-00175-w). URL: <https://doi.org/10.1007/s11721-019-00175-w>.
- [19] Noa Garnett u. a. „Real-Time Category-Based and General Obstacle Detection for Autonomous Driving“. In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV) Workshops*. 2017.
- [20] Fabio Reway u. a. „Test Method for Measuring the Simulation-to-Reality Gap of Camera-based Object Detection Algorithms for Autonomous Driving“. In: *2020 IEEE Intelligent Vehicles Symposium (IV)*. 2020, S. 1249–1256. DOI: [10.1109/IV47402.2020.9304567](https://doi.org/10.1109/IV47402.2020.9304567).
- [21] Anthony Ngo, Max Paul Bauer und Michael Resch. „A Multi-Layered Approach for Measuring the Simulation-to-Reality Gap of Radar Perception for Autonomous Driving“. In: *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*. 2021, S. 4008–4014. DOI: [10.1109/ITSC48978.2021.9564521](https://doi.org/10.1109/ITSC48978.2021.9564521).
- [22] Jonathan Tremblay u. a. „Training Deep Networks With Synthetic Data: Bridging the Reality Gap by Domain Randomization“. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*. 2018.
- [23] Maximilian Denninger u. a. „BlenderProc: Reducing the Reality Gap with Photorealistic Rendering“. In: *Robotics: Science and Systems (RSS)*. 2020. URL: <https://elib.dlr.de/139317/>.
- [24] Jack Collins u. a. „Traversing the Reality Gap via Simulator Tuning“. In: *CoRR* abs/2003.01369 (2020). arXiv: 2003.01369. URL: <https://arxiv.org/abs/2003.01369>.
- [25] Cedric Hartland und Nicolas Bredeche. „Evolutionary Robotics, Anticipation and the Reality Gap“. In: *2006 IEEE International Conference on Robotics and Biomimetics*. 2006, S. 1640–1645. DOI: [10.1109/ROBIO.2006.340190](https://doi.org/10.1109/ROBIO.2006.340190).

- [26] Ekim Yurtsever u. a. „A Survey of Autonomous Driving: Common Practices and Emerging Technologies“. In: *IEEE Access* 8 (2020), S. 58443–58469. DOI: 10.1109/ACCESS.2020.2983149.

## **Selbständigkeitserklärung**

Ich versichere, dass ich die vorliegende Arbeit mit dem Thema:

*„Methoden zur Überwindung der Simulation-to-Reality Gap“*

selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe, insbesondere sind wörtliche oder sinngemäße Zitate als solche gekennzeichnet. Mir ist bekannt, dass Zuwiderhandlung auch nachträglich zur Aberkennung des Abschlusses führen kann.

Leipzig, den 10.03.2023

---

MERLIN FLACH