Fachhochschule Südwestfalen
University of Applied Sciences

Bachelorarbeit

Bewertung und Entwicklung von Algorithmen zur Objekt- und Fahrspurerkennung am Beispiel des autonom fahrenden Duckiebots

Evaluation and development of algorithms for object and lane recognition at the example of the autonomous driving Duckiebot

vorgelegt von

Tim Halle

angestrebter akademischer Grad

Bachelor of Science

Matrikelnummer: 30159345

vorgelegt am: 14. April 2025

Referent: Prof. Dr. Heiner Giefers
Korreferent: Prof. Dr. Michael Rübsam

Zusammenfassung

Die vorliegende Bachelorarbeit widmet sich der Evaluation und Entwicklung von Algorithmen zur Objekt- und Fahrspurerkennung im Kontext autonomer Fahrzeuge am Beispiel des Duckiebots innerhalb der Duckietown-Umgebung. Ziel ist es, verschiedene Ansätze zur Realisierung der autonomen Fahrzeugsteuerung hinsichtlich ihrer Implementierung, Leistungsfähigkeit und Anwendbarkeit zu analysieren und zu vergleichen.

Im Bereich der Objekterkennung wird ein farbbasierter Ansatz auf Grundlage des HSV-Farbraums in Kombination mit den Prinzipien der Braitenberg Fahrzeuge umgesetzt und anhand der Braitenberg Herausforderung aus der Duckietown Learning Experience evaluiert. Für die Fahrspurerkennung werden zwei Reinforcement Learning-Algorithmen (Deep Deterministic Policy Gradient (DDPG) und Twin Delayed DDPG (TD3)) implementiert und sowohl in einer selbst entwickelten Lösung als auch unter Verwendung des Frameworks Stable Baselines 3 miteinander verglichen.

Die Ergebnisse zeigen, dass durch gezielte Anpassung der hyperparameterisierten Implementierungen eine leistungsfähige Spurführung möglich ist. Zudem wird durch den Vergleich mit dem etablierten Framework ein Beitrag zur Bewertung der Leistungsfähigkeit generischer und speziell angepasster reinforcement learning-Lösungen im autonomen Fahrkontext geleistet.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Seit vielen Jahren bietet der öffentliche Straßenverkehr Menschen die Möglichkeit, ihre alltäglichen Wege flexibel und individuell zu gestalten. Weltweit nutzen Millionen tagtäglich Personenkraftwagen (PKW) und andere motorisierte Fahrzeuge, um notwendige Erledigungen zu bewältigen oder entfernte Ziele, etwa im Kontext von Urlaubsreisen, zu erreichen. In Deutschland lässt sich sogar ein wachsender Trend bei der Anzahl zugelassener PKW verzeichnen. Angaben des Kraftfahrt-Bundesamtes belegen, dass die Anzahl der in Deutschland zugelassenen PKW seit 1960 kontinuierlich gestiegen ist. So wurde am 1. Januar 2024 ein Rekordwert von 40 Millionen zugelassenen PKW verzeichnet [1]. Jedoch geht mit diesem Zugewinn an Mobilität und Flexibilität auch ein entsprechendes Risiko einher. So wurden in Deutschland lange immer mehr Straßenverkehrsunfälle verzeichnet. Waren es 1950 etwa 250.000 Unfälle, wurden 2023 bereits etwa 2,5 Millionen gemeldet [2].

Nach Rajasekhar und Jaswal sterben weltweit jedes Jahr etwa 1,24 Millionen Personen an den Folgen von Verkehrsunfällen. Dabei machen 50% der Opfer Personen aus, welche beispielsweise als Fußgänger:in oder Fahrradfahrer:in am öffentlichen Straßenverkehr teilnehmen. In der Altersgruppe wischen 15 und 29 Jahren stellen Unfälle im Straßenverkehr sowie deren Folgen zudem die Haupttodesursache dar [3, S. 1].

Um die Anzahl der Unfälle zu reduzieren und damit auch die Sicherheit für alle Personen zu erhöhen, arbeiten Automobilhersteller sowie Unternehmen aus anderen Branchen daran, Assistenzsysteme für ihre Transportmittel, sowie automatisierte Fahrfunktionen zu entwickeln. Aktuell sind bereits Spurhalteassistenten, automatische Parkassistenten oder auch Kollisionsvermeidung als Assistenzsysteme auf dem Markt verfügbar. Diese Entwicklungen seit den 70er und 80er-Jahren trugen dazu bei, dass die Anzahl der tödlichen Unfälle immer weiter reduziert werden konnte, was sich in den obig erwähnten Statistiken abbildet. So wurden in Deutschland seit der Einführung von ersten Assistenzsystemen bis 2017 immer weniger Verkehrstote festgestellt. Jedoch stagnieren diese Zahlen seit 2017 nahezu und eine Reduzierung ist nur noch in geringem Maße zu erkennen. Tödliche Unfälle im Straßenverkehr lassen sich oft auf menschliches Versagen zurückführen. So wurde festgestellt, dass in 85% der betrachteten Fälle menschliches Versagen erheblichen Einfluss auf Unfälle und deren Folgen hatte. Dass Menschen als Fehlerquelle und Risikofaktor im Straßenverkehr zu betrachten sind, wird bei diesen Zahlen deutlich. Folgerichtig könnten automatisierte Fahrfunktionen oder vollautomatisiert fahrende Fahrzeuge wesentlich zur Erhöhung der Verkehrssicherheit beitragen. Diese Lösungen sind auch Gegenstand der aktuellen Entwicklungen, welche sich darauf konzentrieren, das Fahren mit wachsender maschineller Autonomie zu ermöglichen. Weitere Bestrebungen versuchen diesen Ansatz durch eine Kommunikation zwischen den Fahrzeugen und der Infrastruktur sinnvoll zu ergänzen [4, S. 1011]. Doch nicht nur die Sicherheit könnte spürbar beeinflusst werden. So könnte sich auch in den Bereichen Kraftstoffverbrauch, Verkehrsfluss oder Wirtschaftlichkeit etwa von Logistikunternehmen eine Reduzierung der menschlichen Involvierung positiv auswirken [4, S. 1015]. Die Realisierung solcher autonomen Fahrfunktionen profitiert wesentlich durch den Einsatz von künstlicher Intelligenz (KI) für die Verarbeitung von Sensorinformationen sowie der Fahrzeugsteuerung. So ist künstliche Intelligenz (KI) eine wesentliche Grundlage für Funktionen wie automatisches Ein-

parken des Fahrzeuges, Navigation durch die Umgebung sowie etwaige weitere Entscheidungsprozesse während der Fahrt [5, S. 6].

Derzeit stellt die vollständige Autonomie von Fahrzeugen im öffentlichen Straßenverkehr eine noch ungelöste Herausforderung dar. Zwar bieten einige Automobilhersteller bereits automatisierte Fahrfunktionen an, welche die Fahrenden zeitweise entlasten können, jedoch ist das Fahrsystem immer noch auf einen Menschen hinter dem Lenkrad angewiesen, damit dieser im Falle von Funktionsfehlern eingreifen kann. So sind im Straßenverkehr Systeme zur Erkennung von Hindernissen, Geschwindigkeitsbegrenzungen oder Fahrspuren zu finden. Je nach Hersteller und Alter des betreffenden Fahrzeuges werden unterschiedliche Lösungsansätze genutzt, welche sich in ihrer Zuverlässigkeit unterscheiden. Die Vielzahl an Lösungsmöglichkeiten erschwert Entwickler:innen die Auswahl passender Lösungsansätze sowohl für den Einsatz in PKW als auch in kleineren Modellversuchen im Rahmen eines Proof of Concepts (dt. Beweis des Konzepts). Dies gilt für kommerzielle sowie wissenschaftliche Kontexte.

Um zukünftig einen Vergleich der zahlreichen Lösungsansätze zu erleichtern, werden in der vorliegenden Arbeit Herangehensweisen und Algorithmen zur Erkennung von Objekten und Fahrspuren hinsichtlich ihrer Funktionsweise und Qualität betrachtet. Hierzu werden zunächst theoretische Hintergründe der Lösungsansätze erläutert, um im Anschluss eine Implementierung und Evaluation darstellen zu können. Plattform und Umgebung für diese Betrachtungen bilden der **Duckiebot** sowie die zugehörige virtuelle **Duckietown-Umgebung**. Die erzielten Ergebnisse werden anschließend virtuell evaluiert. Eine Betrachtung der Leistungsfähigkeit in physischen Umgebungen ist im Rahmen dieser Arbeit nicht vorgesehen. Der entstandene Quellcode für die Ergebnisse dieser Arbeit ist in dem GitHub Repository RL-Duckietown[1] zu finden.

1.2 Zielsetzung

Diese Arbeit betrachtet exemplarisch die Herausforderungen der Objekt- und Fahrspurerkennung im Kontext des autonomen Fahrens. Im Bereich der Objekterkennung wird ein Lösungsansatz aus der Duckietown Learning Experience vorgestellt und in der virtuellen Duckietown-Umgebung mittels einer enthaltenen Herausforderung evaluiert. Die Betrachtungen aus dem Bereich der Fahrspurerkennung bilden den Schwerpunkt dieser Arbeit und analysieren die Verwendung von **deep reinforcement learning** (dt. vertiefendes verstärkendes Lernen), kurz DRL, am Beispiel des Duckiebot-Fahrzeuges sowie der virtuellen Duckietown-Umgebung.

Ein solcher Vergleich wird bereits in verschiedenen Arbeiten dargestellt. Beispiel sind Shi et al. [6], welche in ihrer Arbeit verschiedene Algorithmen implementieren und diese miteinander vergleichen. Nach aktuellem Stand des Diskurses wurden solche Vergleiche bisher lediglich auf Grundlage eines reinforcement learning Frameworks oder mittels eigener Implementierungen durchgeführt. Ein Vergleich der Qualität einer Framework-Lösung sowie einer eigens erstellten und auf die Duckietown-Umgebung angepassten Lösung wurde bisher nicht vorgenommen. Die vorliegende Arbeit setzt sich zum Ziel, einen Beitrag zur Schließung dieser Diskurslücke zu leisten und einen Vergleich dieser beiden Optionen herzustellen. Dies erleichtert zukünftig die Abwägung zwischen dem Einsatz einer etablierten Framework Lösung und einer spezifisch auf die Duckietown-Umgebung zugeschnittenen Eigenentwicklung für Anwendungen des reinforcement learnings. Im Detail werden die Algorithmen **Twin Delayed Deep Determenistic Policy Gradient (TD3)** sowie **Deep Determenistic Policy Gradient (DDPG)** auf Grundlage zwei verschiedener Implementierungen verglichen. Diese Implementierungen sind einerseits eine Lösung, welche im Rahmen dieser Arbeit entstanden ist, sowie eine weitere, welche mittels des reinforcement learning Frameworks **Stable Baselines 3** erstellt wurde. Anstelle der Vorgängerversion Stable Baselines wurde aufgrund der großen Beliebtheit sowie der Aktualität des

[1] https://github.com/HalleTim/RL-Duckietown

Frameworks Stable Baselines 3 eingesetzt. Zahlen des Paketmanagers pip zeigen für dieses Abrufe in Höhe von 416.000 Downloads im vorangegangenen Monat [7]. Darüber hinaus wird im entsprechenden GitHub-Repository die Verwendung in über 10.000 weiteren Projekten angegeben [8]. Weiterhin bietet es durch die Unterstützung von geeigneten Simulationsumgebungen die Möglichkeit Duckietown zu implementieren. Dahingegen weist beispielsweise das Framework torchrl lediglich Abrufe in Höhe von etwa 25.000 auf, was die Relevanz in Frage stellen würde [9].

Der Vergleich einer eigens implementierten Lösung mit dem Framework Stable Baselines 3 ist von Interesse, da Stable Baselines 3 bereits vollständige Implementierungen der genannten Algorithmen bietet, welche auf eine Vielzahl von Simulationsumgebungen angewendet werden können. Intuitiv stellt sich daher die Frage, ob und falls ja welcher Qualitätsunterschied zwischen der allgemein anwendbaren Lösung des Frameworks Stable Baselines 3 und der eigens für die Duckietown-Umgebung entwickelten Lösung feststellbar ist. Diese Frage zu beantworten, lässt sich als Ziel der Arbeit definieren. Die für diesen Vergleich verwendeten Metriken werden in Abschnitt 4.5.3 vorgestellt. Für den Bereich der Objekterkennung, werden die bereits in der Herausforderung implementierten Metriken zur Evaluation der Ergebnisse verwendet.

2 Grundlegende Definitionen, Tools und Frameworks

Der nachfolgende Abschnitt führt in zentrale Begriffe und verwendete Strukturen dieser Arbeit ein. Eingangs wird die Terminologie autonomer Fahrzeuge definiert und eine Klassifizierung dieser vorgestellt. Anschließend wird das Duckietown-Projekt sowie der Duckiebot eingeführt, welche in diesem Rahmen als Entwicklungs- und Evaluierungsplattform dienen.

2.1 Definition autonome Fahrzeuge

Unter autonom fahrenden Fahrzeugen werden Fortbewegungsmittel verstanden, welche ihre Insassen selbstständig und sicher zu einem gewünschten Zielort bringen können. Hierbei übernehmen die Fahrzeuge notwendige Aufgaben wie Routenplanung, Navigation und auch die Erfassung der Umgebung [3, S. 1]. Sicherheitsrelevante Aufgaben, wie beispielsweise das Erkennen von Fußgänger:innen oder Hindernissen sowie das entsprechende Ausweichen im Laufe der Fahrt, werden ebenfalls nicht durch einen Menschen ausgeführt.

Die für eine vollständige Autonomie von Fahrzeugen benötigten Funktionen gehören nach Pfischinger und Seiffert [4] zu den kontinuierlich automatisierenden Funktionen. Dies schließt bereits erwähnte Assistenzen wie zur Einhaltung der Fahrspur oder auch Abstandsregulierung zum vorausfahrenden Fahrzeug ein. Jedoch bieten nicht alle automatisierenden Funktionen auch den gleichen Automatisierungsgrad [4, S. 1014]. Beispielsweise automatisiert ein System zur Einhaltung der Fahrspur in geringem Maße im Vergleich mit einem komplexeren System, welches neben der Fahrspur auch die Geschwindigkeit reguliert. Folgerichtig variiert ebenso der Umfang der durch Menschen ausgeführten Aufgaben, anhand des Automatisierungsgrades.

Für eine Gruppierung der Systeme, beziehungsweise Fahrzeuge anhand des Automatisierungsgrades stehen verschiedene, sich aber weitgehend ähnelnde Standards von unterschiedlichen Organisationen zur Verfügung. Beispielsweise verwendet der Verband der Automobilindustrie (VDA) ein fünfstufiges System. Dieses beginnt bei der Automatisierungsstufe Null, welche für eine vollständige Bedienung des Fahrzeuges durch den:die Fahrzeugführer:in steht. Bei der anschließenden Stufe Eins wird dem:der Fahrer:in die Aufgabe der Fahrspureinhaltung oder der Geschwindigkeitsregulierung abgenommen. Entsprechende Assistenzsysteme sind hier in der Lage, dauerhaft eine dieser Aufgaben selbstständig auszuführen. Wenn sowohl die Lenkung als auch die Geschwindigkeit automatisch gesteuert werden können, entspricht ein solches System gemäß VDA der Automatisierungsstufe Zwei. Die Aufgabe des:der Fahrers:in ist hier die Überwachung der Ausführung sowie das Eingreifen im Bedarfsfall. Ist die Aufmerksamkeit des:der Fahrers:in nicht dauerhaft erforderlich, ist dies als Automatisierungsstufe Drei klassifiziert. Die Stufen Vier sowie Fünf sind abschließend in der Lage, Fahrzeuge vollständig autonom zu führen. Unterschiede sind bei diesen beiden Stufen lediglich in den zu unterstützenden Anwendungsfällen zu finden. So können Systeme der Stufe Vier eine autonome Steuerung nur in einigen spezifischen Anwendungsfällen unterstützen [4, S. 1014]. Ein System dieses Entwicklungsstands würde die Fahrzeugsteuerung lediglich auf Autobahnen oder speziell präparierten Streckenabschnitten ermöglichen. Würde dieses Fahrzeug die Autobahn verlassen und in eine Stadt einfahren, wäre ab diesem Moment die Steuerung durch einen Menschen erforderlich. Wenn das Eingreifen eines

Menschen auch in dieser Situation nicht erforderlich ist, kann dies als Stufe Fünf und somit als fahrer:innenloses Fahren verstanden werden [4, S. 1014].

Von vielen Automobilherstellern werden aktuell Systeme der Stufe Zwei angeboten. So verfügen beispielsweise Tesla und Ford über Systeme, welche eine Autonomie der Stufe Zwei ermöglichen. Allerdings liegen in Deutschland bereits auch Genehmigungen für die Einführung von Systemen der Stufe Drei in Fahrzeugen der Marke Mercedes-Benz vor. Nach aktueller Genehmigung dürfen Systeme der Stufe Drei von Mercedes-Benz die Fahrzeugsteuerung bei Geschwindigkeiten bis zu 95km/h übernehmen. Damit wäre dieses aktuell das weltweit am schnellsten fahrende Stufe Drei System in einem Serienfahrzeug [10]. Fahrzeuge der Stufe Vier und Fünf wurden bereits von einigen Unternehmen entwickelt, sind allerdings nicht als Serienfahrzeuge verfügbar und nur in bestimmten Regionen funktional. So werden beispielsweise durch das US-amerikanische Unternehmen Waymo bereits in einigen Städten Taxidienstleistungen durch voll automatisierte Fahrzeuge angeboten [11].

2.2 Duckietown

Für die Entwicklung und Ergebnisevaluierung sind im Rahmen dieser Arbeit die Duckietown-Umgebung sowie der Duckiebot als zentrale Bestandteile anzusehen. Daher wird zunächst das Duckietown-Projekt sowie der zugehörige Duckiebot vorgestellt. Hierbei sollen zentrale Elemente sowie der Aufbau veranschaulicht werden. Darauffolgend wird die **Duckietown Learning Experience** als Element des **MOOC**-Kurses (massive open online course, dt. riesiger offener Online-Kurs) sowie gym-Duckietown als virtuelle Umgebung vorgestellt. Eine genauere Betrachtung der für die Ergebnisse der Spurhaltung verwendeten gym-Duckietown-Umgebung, im Bezug auf die Problemstellung des reinforcement learning erfolgt im Abschnitt 4.2.

2.2.1 Die Duckietown-Umgebung und der Duckiebot

Duckietown ist ein Projekt, welches im Jahr 2016 am MIT (Massachusetts Institute of Technology) ins Leben gerufen wurde, um Studierenden die Herausforderungen autonomer Roboter näher zu bringen. Hierfür sollte eine kleine Plattform geschaffen werden, welche einen einfachen Einstieg in das Fachgebiet der autonomen Roboter bietet und dennoch wissenschaftlichen Ansprüchen in vollem Umfang Rechnung trägt. So entstand das Duckiebot-Fahrzeug in seiner ersten Version, welches bereits 2016 in einem Kurs als Lernträger genutzt werden konnte. In den Folgejahren wuchs die Nachfrage nach dem Duckietbot von anderen Universitäten und Unternehmen, sodass 2018 durch ein Kickstarter-Projekt die Finanzierung der Weiterentwicklung des Duckiebots ermöglicht werden konnte. Qualität, Preisgestaltung sowie Bestellprozesse wurden optimiert und Duckiebots entstanden in sechs Versionen, um Lernenden die Herausforderungen autonomer Roboter beziehungsweise Fahrzeuge näherzubringen, aber auch Forschenden neue Möglichkeiten zu bieten. Zum Zeitpunkt der Erstellung dieser Arbeit im Frühjahr 2025 ist die in Abbildung 2.1 dargestellte Version **DB-21J** die aktuellste auf dem Markt [12].

Ausgestattet ist der DB-21J mit einer inertialen Messeinheit (IMU), einem Hall Sensor, zwei RGB-LEDs sowie einer Kamera und zwei Rädern. Für die Steuerung und Signalverarbeitung wird ein Jetson Nano eingesetzt. Dieser ist durch seine Bauform und Architektur für die Verarbeitung der Sensordaten sowie Ausführung kleinerer KI-Modelle geeignet. Die Stromversorgung aller Komponenten während der Fahrt erfolgt durch eine Batterie mit 10Ah [15].

Für die Evaluierung von Ergebnissen und der Sammlung von Trainingsdaten können sowohl eine physische als auch eine virtuelle Duckietown verwendet werden. Die virtuelle Duckietown, nachfolgend gymDuckietown-Umgebung genannt, kann über das GitHub-Repository [1] abgerufen

[1] https://github.com/duckietown/gym-duckietown

Abbildung 2.1: Duckiebot DB-21J[13]

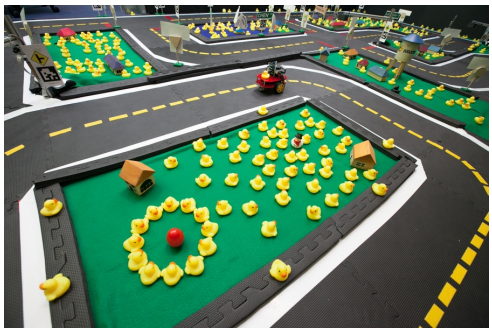


Abbildung 2.2: Duckietown physisch[14]

werden. Diese ist in der Lage unterschiedliche Situationen und Umgebungen in der Duckietown sowie einen virtuellen Duckiebot abzubilden. So kann das Fahrzeug sich sowohl in einem kleinen Kreisverkehr bewegen als auch in einer größeren städtischen Umgebung mit anderen Fahrzeugen sowie Hindernissen. Auf diese Weise können Algorithmen getestet oder KI-Modelle trainiert werden. Nach der Ergebnisevaluation der Daten aus der virtuellen Umgebung kann das Verhalten des Duckiebots auf reale Bedingungen übertragen und hinsichtlich der Qualität geprüft werden. Abbildung 2.2 zeigt eine physische Duckietown mit einigen Verkehrsschildern, Fahrbahnmarkierungen sowie weiteren Komponenten zur Nachbildung der tatsächlichen Herausforderungen des öffentlichen Straßenverkehrs.

Wie leicht zu erkennen ist, entspricht die Duckietown in vielen Punkten nicht der Umgebung im realen Straßenverkehr. So wurden eine Reihe von Abstrahierungen vorgenommen, welche den realen Herausforderungen in vielen Punkten ähneln. Eine Reihe dieser sind in Tabelle 2.1 zu finden.

Reale Welt	**Duckietown-Umgebung**
Fahrzeugsteuerung über Gas- und Bremspedal sowie Lenkrad	Fahrzeugsteuerung über Antrieb des linken oder rechten Rades
einfache Verkehrsschilder	Verkehrsschilder verfügen über QR-Code zur Identifizierung
Fahrbahnmarkierung können dreckig sein und länderübergreifend variieren	Fahrbahnmarkierungen sauber und einheitlich
Verschiedenste Hindernisse	Hindernisse meist Enten oder Pylonen

Tabelle 2.1: Unterschiede zwischen der Duckietown-Umgebung und der realen Welt

2.2.2 Die virtuelle Duckietown-Umgebung

Die virtuelle Duckietown-Umgebung wird für die Entwicklung und Evaluierung von Algorithmen zur Verfügung gestellt. Dies ist eine essenzielle Voraussetzung für die nachfolgenden Betrachtungen, da sie Entwickler:innen die Möglichkeit bietet, über die verfügbaren Hardwarekomponenten hinaus komplexere Szenarien zu simulieren und zu testen. Hierfür werden eine Reihe von Karten zur Verfügung gestellt, welche unterschiedlich komplexe Szenarien abbilden und darüber hinaus modifiziert werden können. Ausschnitte aus der Umgebung sind in Abbildung 2.3 zu sehen.

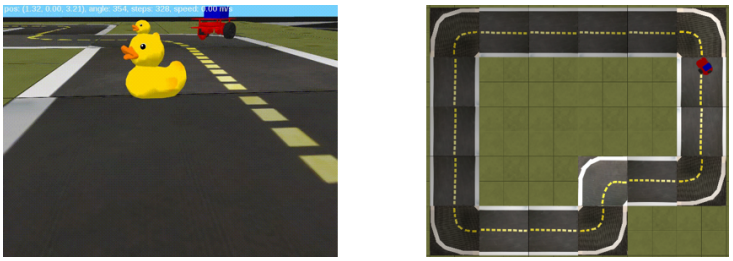


Abbildung 2.3: Beispiel Duckietown-Umgebung[16]

Auch wenn einige Unterschiede zwischen der Umgebung der Learning Experience und gym-Duckietown bestehen, so sind doch die zugrundeliegenden Softwarekomponenten in vielen Teilen identisch. Nachfolgend wird auf einige Besonderheiten dieser Umgebungen eingegangen.

Gym-Duckietown

Gym-Duckietown ist eine mithilfe von OpenGL und Python realisierte Simulationsumgebung, die zur Entwicklung und Evaluation unterschiedlichster Algorithmen für den Duckiebot dient. Die Umgebung bietet verschiedenste Anpassungsmöglichkeiten, wie beispielsweise die Erstellung eigener Karten. Durch die Implementierung des Frameworks OpenAI-gym, ist die Verwendung in Entwicklungen aus verschiedenen Bereichen des machine learning (dt. maschinelles Lernen) möglich. Hierfür werden bereits einige wichtige Funktionen wie ein Belohnungssystem bereitgestellt. Für eine Übertragung der Ergebnisse in die Realität wird beispielsweise eine realitätsnahe Kameraverzerrung implementiert. Zur Verfügung steht die Umgebung in dem bereits erwähnten GitHub-Repository und kann nativ auf einem System oder innerhalb einer Container-Umgebung ausgeführt werden. Eine etwas modifizierte Variante dieser Umgebung wird in der Learning Experience eingesetzt.

Duckietown Learning Experience

Die Duckietown Learning Experience (dt. Lernerfahrung) ist Bestandteil des MOOC, welcher in die theoretischen Hintergründe der autonomen Duckiebots beziehungsweise autonomer Roboter einführt. In Kombination sollen der MOOC und die Learning Experience Lernenden den Einstieg in die Welt der Duckiebots beziehungsweise Duckietown erleichtern und erste Anwendungsszenarien begleiten.

In diesem MOOC werden zunächst mittels einiger Videos und Texte theoretische Hintergründe zu einer konkreten Herausforderung der Autonomie vermittelt. Dies kann z.B. die Objekt- oder Hinderniserkennung sein, welche in diesem Kurs mittels der Theorie von **Braitenberg Fahrzeugen** oder auch mittels Methoden aus dem Bereich des **deep learning** (dt. vertiefendes Lernen) abgebildet werden. Nach der Wissensvermittlung kann das Gelernte in der Duckietown Learning Experience angewendet werden. Diese bietet für die Anwendung der theoretischen Inhalte jeweils passende praktische Übungen im Rahmen einiger Herausforderungen, welche zunächst in der virtuellen Duckietown-Umgebung stattfinden. Hier wird beispielsweise zu der Objekt- und Hinderniserkennung die Braitenberg Herausforderung angeboten, welche zur Aufgabe hat, den Duckiebot ein Feld mit vielen Enten ohne Kollisionen passieren zu lassen. Hierbei wird der:die Lernende auf dem Weg zu Lösungsfindung mittels einiger Jupyter Notebooks begleitet. Für die Evaluation der Ergebnisse stellt die Herausforderung eine eigene Metrik zur Verfügung, welche lokal auf dem Endgerät des:der Nutzers:in berechnet werden kann. Alternativ ist die Verwendung einer serverseitige Evaluation für eine Platzierung auf der öffentlichen

Bestenliste möglich. Einige Herausforderungen erlauben durch die Übertragung der Lösung auf den physischen Duckiebot einen Vergleich von Simulation und Realität.

Im Rahmen dieser Arbeit wird die Braitenberg Herausforderung für eine Einführung in das Feld der Objekterkennung sowie den Duckiebot behandelt. Da die Aspekte und Ergebnisse des reinforcement learnings nicht Bestandteil der Learning Experience sind, werden diese Ergebnisse unter abweichenden Gesichtspunkten betrachtet. Die Evaluation erfolgt anschließend lediglich in der virtuellen Umgebung. Ein Vergleich zu Ergebnissen unter realen Bedingung ist ausdrücklich nicht Bestandteil dieser Arbeit und bildet somit ein Forschungsdesiderat.

Für die Ausführung der Herausforderungen wird auf dem lokalen Computer der lernenden Person ein Docker-Container ausgeführt, welcher die Umgebung der jeweiligen Learning Experience bereitstellt. Basisimage der Container ist duckietown/dt-commons[2], welches über DockerHub abgerufen werden kann. Die Docker-Container führen die Umgebung der Learning Experience, die Herausforderung sowie die Berechnung der Metrik für die Bestimmung des Gesamtergebnisses durch. Hierbei werden im Vorfeld von den Entwickler:innen festgelegte Karten verwendet.

2.2.3 Hard- und Softwareseitige Rahmenbedingungen

Die Ausführung des Trainings erfolgte bei allen der nachfolgend betrachteten Ergebnisse auf einem lokal betriebenen Ubuntu System. Hierdurch konnte die Vergleichbarkeit der Ergebnisse, insbesondere hinsichtlich der Laufzeit des Trainingsprozesses, gewährleistet werden. Die Berechnung der nachfolgend betrachteten **neuronalen Netze** erfolgte mittels einer CUDA-fähigen Grafikkarte. Die zentralen Systemkomponenten auf Hard- und Softwareebene sind nachfolgend aufgelistet.

- CPU: AMD Ryzen 7 3700x - 3,6GHz - 8 Kerne
- GPU: Nvidia Geforce RTX 4070Ti - 12GB GDDR6
- RAM: 32 GB DDR4
- OS: Ubuntu 22.04 Desktop
- Python: Version 3.8.20
- Pytorch: Version 2.4.1

[2]https://hub.docker.com/r/duckietown/dt-commons

3 Objekterkennung

Die Objekt- beziehungsweise Hinderniserkennung wird nachfolgend am Beispiel der Braitenberg Herausforderung erläutert. In dieser Umgebung weicht das Fahrzeug Objekten aus, die es anhand ihrer Farbe visuell erfasst. Dafür werden Grundlagen der Braitenberg Fahrzeuge sowie eine Erkennung der Objekte mittels des **HSV-Farbraum** verwendet. Um das Verständnis der Lösung nachfolgend zu erleichtern, erfolgt zunächst die Einführung in einige theoretische Grundlagen. So werden die Braitenberg Fahrzeuge 1 bis 3 als Grundlage der Fahrzeugsteuerung vorgestellt sowie anschließend die Objekterkennung im HSV-Farbraum. Darauffolgend werden das Ziel der Herausforderung sowie ein im Rahmen dieser Arbeit entstandener Lösungsansatz vorgestellt. Abschließend erfolgt die Evaluation der Ergebnisse mit einer Einordnung in die aktuell verfügbare Bestenliste.

3.1 Die Braitenberg Fahrzeuge

Valentin Braitenberg ist Begründer der sogenannten Braitenberg Fahrzeuge. Er setzte sich intensiv mit den Strukturen von Tiergehirnen auseinander und versuchte diese in Form von Maschinen nachzubilden [17, S. 1]. Inspiriert von der evolutionären Entwicklung dieser Tiere, entwickelte er die Fahrzeuge (auch Vehikel genannt) 1 bis 14. Diese verfügen über verschiedene Sensoren, ein oder zwei Räder mit jeweils einem Motor und teilweise auch über ein Stützrad zur Gewährleistung der Stabilität. Die Sensoren werden mit den antreibenden Motoren verbunden und steuern diese bei Erfassung von Messwerten an [18, S. 2]. Trotz eines grundlegend gleichen Aufbaus variieren die Vehikel in ihrer Komplexität. So ist **Vehikel 1** als erste und somit einfachste Variante zu verstehen [17, S. 2]. Wie in Abbildung 3.1 zu sehen, besteht Vehikel 1 lediglich aus einem rückseitig montierten Rad mit Motor sowie einem Sensor an der Front. Die Stärke des durch den Sensor gemessenen Reizes entscheidet darüber, wie schnell sich das Fahrzeug in Richtung des Pfeiles bewegt. Allerdings ist es nicht in der Lage, Lenkbefehle zur Kursänderung auszuführen [17, S. 4]. Mittels dieses Konzeptes könnte beispielsweise ein einfaches Fahrzeug konstruiert werden, welches sich bei Sonneneinstrahlung bewegt und bei Dunkelheit abschaltet beziehungsweise anhält.

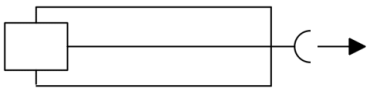
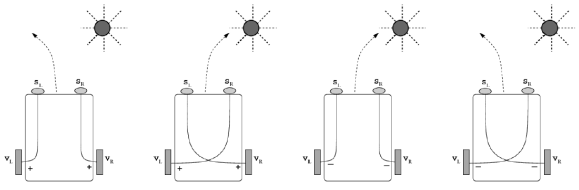


Abbildung 3.1: Vehikel eins[17, S. 4]



(a) Vehicle 2a (b) Vehicle 2b (c) Vehicle 3a (d) Vehicle 3b

Abbildung 3.2: Varianten Vehikel zwei und drei[19, S. 2]

Komplexere Bewegungsmuster wie das Navigieren zu einer Lichtquelle mittels Richtungsänderung erfordern die Verwendung der weiterentwickelten Fahrzeuge. Zunächst soll **Vehikel 2** näher betrachtet werden, da es unmittelbar auf den Grundlagen des ersten Vehikels aufbaut. Dieses Fahrzeug ist im hinteren Teil mit zwei Rädern, die jeweils über einen eigenen Motor

verfügen ausgestattet, sowie an der Frontseite mit Sensoren auf der linken und rechten Seite. Folgerichtig gibt es hier nun verschiedene Möglichkeiten, Sensoren mit Motoren zu verbinden. Da diese Paarungen entscheidend für das Fahrzeugverhalten sind, unterteilte Braitenberg Vehikel 2 in 2a und 2b für eine leichtere Unterscheidung. **Vehikel 2a** verbindet die Sensoren mit den Motoren der jeweils gleichen Seite. Dies führt zu einem Verhalten, welches der Autor als fear (dt. Angst) beschreibt [17, S. 8f]. In Abbildung 3.2(a) kann das entstehende Verhalten am Beispiel eines Fahrzeuges mit Lichtsensoren beobachtet werden. Wie leicht zu erkennen ist, neigt das Fahrzeug dazu, sich von der Lichtquelle zu entfernen und ist bestrebt, nicht mit dieser in Kontakt zu kommen. Grund hierfür ist, dass in diesem Beispiel durch den Lichteinfall von rechts der rechte Sensor eine höhere Helligkeit messen wird als der linke. Infolgedessen wird das rechte Rad mehr beschleunigen als das linke, wodurch eine Linkskurve ausgelöst und der Lichtquelle ausgewichen wird. Ein Sonderfall stellt die Positionierung der Reizquelle exakt in der Mitte beider Sensoren dar. In diesem Fall würden beide Sensoren gleichermaßen gereizt werden und das Fahrzeug sich daher auf die Quelle zu bewegen [17, S. 9].

Vehikel 2b verhält sich gegensätzlich zu 2a und steuert direkt auf die Reizquelle zu. Die Sensoren steuern nun den Motor auf der jeweils gegenüberliegenden Seite an, was dazu führt, dass das Fahrzeug bei Annäherung an die Reizquelle zunehmend beschleunigt wird. Braitenberg selbst beschreibt dies als aggressives Verhalten, da der Eindruck entsteht, dass das Fahrzeug die Quelle zerstören möchte [17, S. 9]. Abbildung 3.2(b) zeigt dieses Verhalten erneut durch eine Lichtquelle. Wäre die Lichtquelle hier eine Stehlampe, so wäre es durchaus denkbar, dass das Fahrzeug diese durch sein aggressives Verhalten umwirft und zerstört. Zur Vollständigkeit sei erwähnt, dass offensichtlich auch beide Sensoren mit beiden Motoren verbunden sein können (Vehikel 2c). Das Verhalten entspricht in diesem Fall exakt dem von Vehikel 1 und ist aus diesem Grund zu vernachlässigen [17, S. 8].

Bei den bisher vorgestellten Fahrzeugen führt die hohe Geschwindigkeit zu einer intensiven Stimulierung der Sensoren. In Abbildung 3.2 (a) und (b) wird dies auch durch ein + auf den Verbindungen kenntlich gemacht. Bei **Vehikel 3** führen, bei gleichem Aufbau des Fahrzeuges starke Stimuli zu einer Reduzierung der Geschwindigkeit. Dadurch wird **Vehikel 3a** wie auch in Abbildung 3.2 (c), sich langsam der Reizquelle zuwenden, bis es bei ausreichender Nähe zum Halten kommt. **Vehikel 3b** hingegen wendet sich von der Reizquelle ab und sucht weiter nach noch stärkeren Reizen. Daher nennt Braitenberg dieses Fahrzeug auch Entdecker [17, S. 12].

Zusammenfassend ist das Verhalten der Vehikel 2a und 3b beziehungsweise 2b und 3a ähnlich. Unterscheidbar macht sie die Auswirkung von einem hohen Stimulus der Sensoren, welcher Vehikel 2 beschleunigen und Vehikel 3 verlangsamen lässt [19, S. 2]. Obwohl die Konstruktion dieser Fahrzeuge denkbar einfach ist, können einfache Verhalten von Tieren sowie Fahrzeugen dargestellt werden [19, S. 1]. Es könnte sogar der Eindruck entstehen, dass das Fahrzeugverhalten einer Form von Wissen ähnelt. Dies ist jedoch ein Trugschluss, da das System des Fahrzeuges lediglich tierähnliche Verhaltensweisen erzeugt [17, S. 14].

3.2 Objekterkennung im HSV-Farbraum

Im vorangegangenen Abschnitt wurde dargestellt, welche einfachen Grundprinzipien verwendet werden können, um mittels Messwerten von Sensoren ein Fahrzeug zu steuern. Allerdings war in den genannten Beispielen zur Erkennung der Reizquelle keine Vorverarbeitung der Daten erforderlich, da beispielsweise der Lichtsensor direkt für die Erkennung der Lichtquelle konzipiert ist. Die im nachfolgenden Abschnitt thematisierte Herausforderung setzt eine Kamera als Bildsensor ein, um Reizquellen auszumachen. Da offensichtlicherweise die Kamera neben den Hindernissen auch eine Reihe von irrelevanten Reizen aufzeichnen wird, gilt es zunächst die interessanten auf dem Bild ausfindig zu machen. Um dies zu tun, gibt es verschiedene Möglichkeiten. So kann

mittels Methoden der KI ein Modell trainiert werden, welches in der Lage ist bekannte Objekte in einem Bild zu markieren. Hassan, Ming und Wah [20] schreiben hier, dass diese Techniken insbesondere Probleme bei der Erkennung kleinerer Objekte aufzeigt. Dies liegt nicht zuletzt an der starken Auflösungsreduzierung bei der Bildverarbeitung. So ist es üblich, dass Bilder mit einer originalen Auflösung von 4096x3072 Pixeln bei der Eingabe in das KI-Modell auf eine sehr kleine Auflösung wie beispielsweise 300x300 Pixel reduziert werden. Diese starke Reduktion macht das Auffinden kleinerer Objekte zu einer sehr großen Herausforderung, da nun noch weniger Pixel für die Darstellung des ohnehin kleinen Objektes verwendet werden [20, S. 1]. Das Hervorheben der Farbe gesuchter Objekte kann die Suche allerdings erleichtern. Beispielsweise wäre es denkbar, zum Auffinden einer grünen Fußgängerampel alle grünen Objekte in einem Bild hervorzuheben und den Rest auszublenden. Dies würde das Auffinden für die KI vereinfachen, da die Bildinformationen drastisch reduziert werden [20, S. 2]. Diese Markierung von Objekten basierend auf ihrer Farbe, wird auch top down-Ansatz genannt [21, S. 1]. Hierfür wird häufig eine binäre Maske verwendet, welche die Farbinformationen des Bildes auf schwarz und weiß reduziert. Ziel hierbei ist es, die überflüssigen Informationen zu entfernen, welche für die Verarbeitung nicht länger benötigt werden. Die Entscheidung über relevant oder irrelevant wird anhand einer Ober- und Untergrenze des Farbbereiches getroffen. Alle Pixel, welche nicht in diesen Farbbereich fallen, werden geschwärzt und somit aus den Bildinformationen gelöscht [22, S. 1].

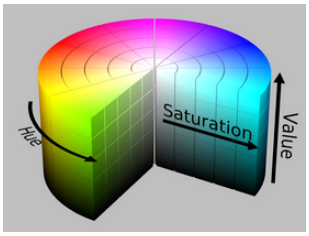


Abbildung 3.3: HSV-Farbraum[23]

Abbildung 3.4: Anwendung binäre Maske für grüne Objekte[20, S. 2]

Die Farben eines Bildes sind durch die Menge der Farbwerte in den vorhandenen Pixeln definiert. Jeder dieser Pixel hat einen eigenen Farbwert, welcher bei der Anzeige auf einem Monitor dargestellt wird. Die Kameras, welche die Bilder aufnehmen, verwenden häufig die Mischung von rot, grün und blau, um die Farbwerte der einzelnen Pixel zu speichern. Dieser so entstehende Farbwert wird, aufgrund der verwendeten Basisfarben **RGB** genannt. Die Intensität der einzelnen Farben wird je mit einem Wert zwischen null und 255 angeben. Durch die Mischung aller drei Farbintensitäten ergibt sich die endgültige Zielfarbe des Pixels [22, S. 1]. Allerdings ist diese Angabe bei der Suche nach Farben in Bildern häufig problematisch. Zum einen ist ein RGB-Wert für Menschen nicht intuitiv zu verstehen und es fällt nicht immer leicht zu erkennen, welche Farbe ein gegebener Wert repräsentiert [20, S. 2]. Weiterhin kann es bei der Verwendung von RGB während der Objektsuche zu Problemen kommen, sollten Veränderungen der Helligkeiten auftreten. Aus diesen Gründen werden die Bilder für die Bildverarbeitung meist in das **HSV-Format** übertragen [22, S. 1]. Dieses Format wurde 1970 mit dem Ziel entwickelt das menschliche Verständnis von Farben besser abzubilden [20, S. 2]. Ebenso ist die Handhabung von Helligkeitsänderungen leichter und auch die Angabe von Farbwerten fällt Menschen einfacher [22, S. 1]. Das HSV-Format besteht aus den Komponenten hue, saturation und value (dt. Farbe, Sättigung und Wert). Diese drei Werte werden, wie auch in Abbildung 3.3 zu sehen, in einem zylindrischen Körper angeordnet. Hue ist die verwendete Farbe und wird mittels eines Wertes zwischen 0 und 360 Grad auf der Zylinderkreisbahn angegeben. Saturation gibt die Intensität der Farbe an, welche durch eine prozentuale Angabe zwischen null und 100% angegeben wird.

Die Helligkeit ist repräsentiert durch value, welche ebenso eine prozentuale Angabe von null bis 100% verwendet. Mittels dieser Skala kann nun eine Ober- und Untergrenze für die binäre Maske definiert werden, um die unerwünschten Informationen zu entfernen [24, S. 1]. Ein Beispiel für die Anwendung auf Objekte mit grünen Farben ist in Abbildung 3.4 zu sehen. Ursprüngliches Ziel hier war es die grüne Fußgängerampel zu finden. Hier könnte für hue ein Bereich zwischen 155° und 160° gewählt werden, um verschiedene Grüntöne in die Auswahl einzuschließen. Ähnliche Bereiche sind durch entsprechende Prozentwerte für saturation und value anzugeben. Wie in dem Bild zu sehen, verbleiben nach Anwendung der Maske ausschließlich die gewünschten Farbbereiche im Bild. Jedoch hat dies ebenso zur Folge, dass unerwünschte Objekte wie die Mülltonne und das Straßenschild im Bild enthalten bleiben. Gelegentlich kommt es auch vor, dass einzelne Bildbereiche ein Rauschen aufweisen, da ihre Farbgebung nahe an den gewählten Grenzbereichen liegt. Dieses kann durch einen Algorithmus, welcher die mittleren Farbwerte benachbarter Pixel verwendet, verhindert werden [25, S. 2].

3.3 Die Braitenberg Herausforderung

Wie bereits Eingangs erläutert, werden im Rahmen des MOOC der lernenden Person für die praktische Anwendung des Gelernten Herausforderungen gestellt. Diese sind Bestandteil der Duckietown Learning Experience, welche unterschiedliche Themenfelder des autonomen Fahrens mit dem Duckiebot abdeckt. Die Herausforderungen werden dabei in einer virtuellen Umgebung absolviert, welche das Fahrzeug je nach Herausforderung in eine Umgebung mit verschiedenen Hindernissen oder in einen Stadtkurs bringt. Die lernende Person hat jeweils zur Aufgabe einen Algorithmus zu entwickeln, welcher das Fahrzeug in die Lage versetzt, durch die virtuelle Umgebung zu navigieren. Der Erfolg beziehungsweise Misserfolg wird durch die Duckietown-Umgebung bewertet. Diese gibt abschließend auch ein Feedback über die Qualität der geprüften Lösung.

Die Braitenberg Herausforderung hat zur Aufgabe, den Duckiebot durch ein Feld mit zufällig platzierten gelben Enten zu navigieren. Die gelben Enten stellen Hindernisse dar, welchen das Fahrzeug für ein erfolgreiches Bestehen ausweichen muss. Hierfür ist ein passender Algorithmus für die Fahrzeugsteuerung zu entwickeln, welcher die Enten auf dem Kamerabild erkennt und diese mit den Grundprinzipien der Braitenberg Fahrzeuge umfährt. Wird eine Kollision mit einer der Enten festgestellt, gilt die Herausforderung sofort als gescheitert. Das Feld gilt als erfolgreich passiert, wenn das Fahrzeug am Ende des Feldes angekommen ist und keine Berührungen stattgefunden haben. Während der Simulationszeit werden verschiedene Statistiken durch die Ausführungsumgebung erstellt. So wird nach Abschluss des Durchlaufs eine zusammenfassende Statistik angezeigt, in welcher beispielsweise zu sehen ist, wie lange das Fahrzeug gefahren ist und welche Distanz es in dieser Zeit zurückgelegt hat. Da das Passieren eines Feldes nicht immer ausschlaggebend für die Qualität der Lösung ist, sind innerhalb der Simulation fünf dieser Entenfelder zu durchfahren. Die Ergebnisse werden abschließend kumuliert und in einem zentralen Ergebnisbericht zusammengefasst. Diese Erstellung des Ergebnisberichts kann lokal auf dem Computer der lernenden Person erfolgen oder online auf den durch Duckietown bereitgestellten Servern. Bei einer Evaluation auf den Duckietown-Servern werden neben den numerischen Metriken auch visuell die zurückgelegte Wegstrecke des Fahrzeuges bei allen fünf Feldern angezeigt. Abbildung 3.5 zeigt beispielhaft die zurückgelegte Strecke eines Fahrzeuges auf der Karte sowie die verarbeitete Aufnahme der Frontkamera. Die rechte Abbildung zeigt das Feld, welches das Fahrzeug durchfährt mit den durch gelbe Punkte repräsentierten Enten. Die diagonal verlaufende blaue Linie stellt die gefahrene Strecke des Fahrzeuges dar.

Auf der bereits erwähnten Kamera des Fahrzeuges sind nun zunächst die Reizquellen zu identifizieren, welche den Lenkimpuls des Fahrzeuges beeinflussen. Da die im Rahmen der Her-

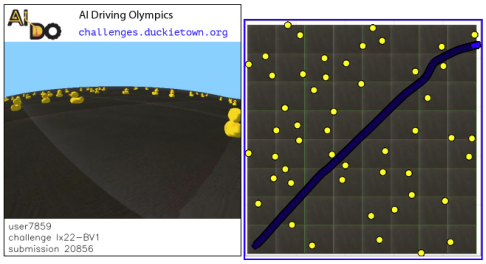


Abbildung 3.5: Braitenberg Herausforderung[26]

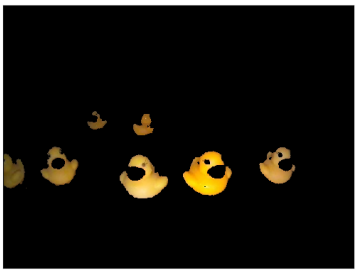


Abbildung 3.6: Enten maskiert

ausforderung zu vermeidenden Objekte grundsätzlich gelb gefärbt sind, ist eine farbbasierte Objekterkennung realisierbar. Hierfür wurde, wie im vorherigen Abschnitt erläutert, eine binäre Maske erstellt, welche die Enten beziehungsweise gelben Bildbereiche markiert. Die für diesen Zweck verwendete Bibliothek ist Open-CV. Diese wandelt zunächst das im RGB-Farbformat aufgenommene Bild in das HSV-Format um. Anschließend werden die Farben mittels der zuvor definierten Werte für die Ober- und Untergrenze maskiert beziehungsweise gefiltert. Für die Bestimmung der Grenzwerte wurden Bilder von gelben Enten unter realen Lichtbedingungen verwendet, um bei einer eventuellen Adaption auf einen physischen Duckiebot verwendbare Ergebnisse zu erzielen. Durch dieses Vorgehen ist der relevante Farbbereich deutlich größer als bei der Beschränkung auf Referenzen aus der Simulation. Wie in Abbildung 3.5 zu sehen, treten in der virtuellen Umgebung lediglich wenige Schatteneffekte auf. Unter realen Bedingungen, wie auch in Abbildung 3.6 zu sehen, würden weitere Abstufungen der Farben auftreten, was gegebenenfalls das Ergebnis beeinflussen könnte. So sind in der Abbildung die Enten in unterschiedlichen Gelbtönen zu sehen. Für die Bestimmung der HSV-Farbwerte wurde mittels eines Farbwählers an verschiedenen Stellen auf den Enten der Farbwert ermittelt. Auf diese Weise konnten Maxima und Minima für die Farbgebung der Enten auf dem Bild festgelegt werden. Für eine Anpassung an unterschiedliche Situationen und Lichtverhältnisse wurde dieses Vorgehen mit verschiedenen Beispielbilder aus der Learning Experience durchgeführt. Abbildung 3.6 zeigt das Ergebnis nach Anwendung der Maske auf die Kamerabilder. Hier ist zu sehen, dass die Enten in weiten Teilen erhalten bleiben. Jedoch sind an einige Stellen die Enten nicht vollständig dargestellt und insbesondere im Hintergrund weisen die Enten unter dunkleren Lichtverhältnissen schwarze Stellen auf.

Das so entstehende Bild hebt nun lediglich die Bereiche hervor, welche für das Fahrzeug einen entsprechenden Reiz im Sinne der Braitenberg Fahrzeuge darstellt. Wie im Abschnitt 3.1 erläutert, verwenden die Vehikel 2 und 3 insgesamt zwei Sensoren, welche jeweils für die Ansteuerung eines Rades zuständig sind. Um dieses Ergebnis mittels des maskierten Kamerabildes zu erhalten, ist eine Aufteilung des Bildes in zwei sensorisch sensitive Bereiche erforderlich. Diese Bereiche lösen eine Beschleunigung des linken oder rechten Rades des Fahrzeuges aus, sollte eine der gelben Enten im entsprechenden Bereich auftauchen, um dieser auszuweichen. Hierfür werden die Pixel des maskierten Bildes als Matrix der Farbwerte verwendet. Folgerichtig sind die Bereiche, welche mit schwarz gefüllt sind, mit dem Farbwert null versehen. Gelbe Bereiche hingegen weisen größere Zahlen auf, welche zur Ansteuerung der Motoren verwendet werden können. Für die Intensität der Motorenansteuerung wird je Motor eine Matrix zur Festlegung der sensorischen Bereiche verwendet. Diese weist die gleichen Dimensionen wie das maskierte Bild auf und hat an den Positionen, welche einen sensorischen Reiz für den Vortrieb des linken beziehungsweise rechten Motors darstellen, den Wert eins. Alle weiteren sensorisch nicht relevanten Positionen werden in der Matrix mit dem Wert null versehen.

Listing 3.1: Formel Motorleistung

```
1         left_motor  = const + gain * np.sum(LEFT * preprocess(image) )
2         right_motor = const + gain * np.sum(RIGHT * preprocess(image) )
```

Erfolgt die Bestimmung der sensitiven Bereiche nach den Prinzipien von Braitenberg Vehikel 2a, ist um den Enten auszuweichen, der linke Motor durch die Präsenz von Enten in der linken Bildhälfte anzusteuern. Für den rechten Motor gilt parallel selbiges Prinzip. Mittels Multiplikation der beschriebenen Matrizen für den Vortrieb und des maskierten Bildes wird die Bildmatrix für den entsprechenden Motor in verschiedene reizvolle Bereiche reduziert. Wie in dem Listing 3.1 zu sehen ist, werden die Werte der resultierende Matrix zur Berechnung, eines skalaren Wertes addiert und mit dem Parameter gain zur Anpassung des Skalar multipliziert. Der Parameter const stellt die konstante Geschwindigkeit dar, mit welcher sich der Motor auch ohne Reizbeeinflussung drehen würde, damit sich das Fahrzeug mit gleichbleibender Geschwindigkeit vorwärts bewegt. Als Ergebnis dieser Formel wird die Geschwindigkeit des linken beziehungsweise rechten Motors in Abhängigkeit von gelber Farbe im entsprechenden jeweils relevanten Bildabschnitt berechnet. Auf diese Weise kann nun das Fahrzeug selbstständig durch das Feld der Enten navigieren.

3.4 Evaluation der Ergebnisse

Die Ergebnisse dieser Arbeit setzen die Strukturen des Braitenberg Fahrzeuges 2a um. Dieses ermöglicht dem Duckiebot durch das in Abschnitt 3.1 beschriebene Verhalten den Enten auszuweichen. Sowohl Fahrzeug 2b als auch 3a wurden nicht für eine Verwendung in Betracht gezogen, da diese der Objektannäherung dienen. Nachfolgend werden die erzielten Ergebnisse auf Grundlage des Fahrzeuges 2a dargestellt.

Die Umsetzung der sensorischen Bereiche innerhalb dieser Arbeit ist in Abbildung 3.7 dargestellt. Diese visualisiert zunächst das Ergebnis nach Anwendung der binären Maske an einem beispielhaften Bild. Darauffolgend ist in den Matrizen left wheel (dt. linkes Rad) und right wheel (dt. rechtes Rad) jeweils in rot zu sehen, welche Bereiche des Bildes eine zusätzliche Beschleunigung des jeweiligen Rades auslösen. Wie im vorherigen Abschnitt bereits erläutert, erfolgt die Reduzierung auf sensorische Bereiche durch die Verwendung einer entsprechend definierten sensorischen Matrix. So sind in den jeweiligen Matrizen der Räder die Bestandteile der Enten dargestellt, welche das linke beziehungsweise rechte Rad beschleunigen. Für eine Fokussierung auf die Hindernisse direkt vor dem Fahrzeug werden, wie insbesondere in der linken Matrix zu sehen, Objekte im Hintergrund nicht für die Motoransteuerung berücksichtigt. Auf diese Weise kann das Fahrzeug Hindernissen ausweichen, mit welchen sonst eine zeitnahe Kollision erfolgen würde. Weiterhin ist festzustellen, dass bei der Ansteuerung des linken Rades ein großer Bestandteil des selbigen Bildrandes nicht sensorisch erfasst wird, wohingegen bei der Ansteuerung des rechten Rades in Richtung der x-Achse alle Enten erfasst wurden. Dieser Umstand verdeutlicht die verwendete Asymmetrie zwischen der linken und rechten sensorischen Matrix. Hierdurch wird der in Abschnitt 3.1 beschriebene Sonderfall umgangen, der auftritt, wenn sich Objekte mittig im sensorischen Bereich aufhalten. Befinden sich nun Objekte in der Mitte des Sichtfeldes, weicht das Fahrzeug diesen durch eine stärkere Ansteuerung des rechten Motors entsprechend aus.

Um die Vergleichbarkeit der Ergebnisse dieser Arbeit zu gewährleisten, wurde die dargestellte Lösung durch den offiziellen Duckietown-Server evaluiert. Wie bereits zu Beginn erwähnt, führt dieser eine Auswertung sowie eine Rangliste der eingereichten Lösungen aller Nutzenden. Gelistet sind die nachfolgend beschriebenen Ergebnisse unter der Nummer 28732[1] und der Nutzernummer

[1] https://challenges.duckietown.org/v4/humans/submissions/28732

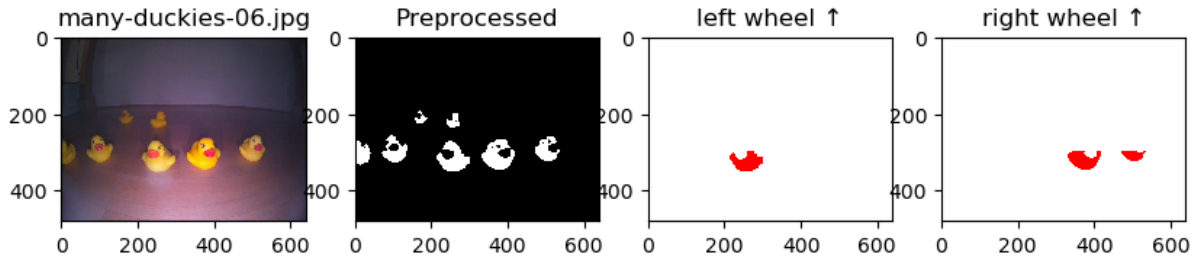


Abbildung 3.7: Ergebnis der Objekterfassung und Motorbeschleunigung

9667. Grundlage der Evaluierung ist die Ausführung des Algorithmus in fünf unterschiedlichen Simulationen. Die in diesen gefahrene Distanz wird gemittelt und ist der entscheidenden Parameter für die Einordnung auf der Bestenliste. Bei der Betrachtung der Ergebnisdaten ist in allen fünf Simulationen ein kurzfristiges Ausweichen des Fahrzeuges vor Eintreten einer Kollision mit einem Hindernis festzustellen. Besonders markant ist dies, sobald sich eine Ente mittig vor dem Fahrzeug befindet. In Simulation d40 führt dieses Verhalten zu einer Kollision, wie durch die dargestellte Route zu sehen ist. Potenziell vermeidbar wäre dieses Verhalten durch eine Erhöhung des Parameters gain in der im vorherigen Abschnitt beschriebenen Gleichung. Jedoch führte eine Erhöhung des Wertes zu einem sehr starken Einlenkverhalten, welches in der Folge vermehrt Kollisionen mit Enten auslöste. Die verbleibenden vier Simulationen wurden erfolgreich durchlaufen. Jedoch ist zu erkennen, dass nicht immer die Diagonale des quadratischen Feldes für die größtmögliche Distanz gewählt wurde. Aus diesem Grund beträgt der mittlere Weg des Fahrzeuges im Rahmen dieser Lösung lediglich 4,9 Meter. Durch weitere Optimierungen der Parameter könnte eine Verfeinerung der Reaktion des Fahrzeuges erreicht werden, um die gefahrene Distanz weiter zu steigern.

4 Fahrspurerkennung

Der positive Einfluss von KI auf die Qualität und die Weiterentwicklung autonomer Assistenzsysteme wurde bereits einleitend in dieser Arbeit dargestellt. Um eine KI in die Lage zu versetzen, eine spezifische Aufgabe durchzuführen, werden in der Regel große Datenmengen für den Trainingsprozess benötigt. Die Verfügbarkeit dieser Daten ist im Automobilbereich häufig problematisch, was die Entwicklung und das Training der Modelle erschwert. Aus diesem Grund sind viele Unternehmen bestrebt, die für das Training benötigte Datenmenge zu reduzieren oder alternativ den Datenbestand zu vergrößern. Zur Lösung der Problematik des Datenbestandes kann der Einsatz von **reinforcement learning** (dt. verstärkendes Lernen) beitragen, da diese Verfahren in der Lage sind, durch Simulationsumgebungen und spezielle Algorithmen KI-Modelle zu trainieren [27, S. 2]. Im nachfolgenden Kapitel werden reinforcement learning Algorithmen am Beispiel der Duckietown-Umgebung sowie des Duckiebots betrachtet. Hierbei werden zunächst Grundlagen und Prinzipien des reinforcement learning erörtert, bevor anschließend DDPG (deep determenistic policy gradient, dt. tiefe deterministische Regelwerk Gradienten) sowie TD3 (twin delayed DDPG, dt. Zwillings verzögertes DDPG) als Vertreter der reinforcement learning-Algorithmen vorgestellt und innerhalb des Duckietown Frameworks evaluiert werden. Die Betrachtung hinsichtlich dieser Algorithmen sind aufgrund ihrer Fähigkeiten in Bezug auf die Handhabung hoch dimensionaler Aktionsräume sowie Zustände relevant [28, S. 2]. Da diese in der Duckietown-Umgebung vorliegen können sowohl TD3 als auch DDPG als geeignet erachtet werden. Die Betrachtungen erstrecken sich dabei über eine Implementierung mittels Stable Baselines 3 sowie eine im Rahmen dieser Arbeit entstandenen Implementierung.

4.1 Reinforcement learning

Die Interaktion zwischen einem **Agent** und einer **Simulationsumgebung** bilden die Grundlage für die Realisierung von reinforcement learning. Repräsentiert werden diese Komponenten in der vorliegenden Arbeit durch das virtuelle Duckiebot-Fahrzeug (Agent) sowie die virtuelle Duckietown-Umgebung (Simulationsumgebung), durch welche sich der Agent bewegt. Die Interaktion von Agent und Umgebung ist dabei geprägt durch den iterativen Informationsaustausch von Aktionen des Agenten (action) sowie **Belohnungen** (reward) und **Zustandsinformationen** (state) aus der Umgebung.

Der in Abbildung 4.1 dargestellte iterative Zyklus stellt die grundlegende Interaktion von Agent und Simulationsumgebung während des Trainingsprozesses dar. Das Bestreben des Agenten ist es, während dieses Zyklus unter Einsatz der Zustandsinformationen sowie der Belohnung eine bestimmte Aufgabe zu erfüllen und die Summe der erhaltenen Belohnung zu maximieren [29, S. 6]. Hierfür versucht er während des nachfolgend dargestellten Prozesses, eine **policy** (dt. Regelwerk) zu entwickeln, welche für die Bestimmung der im nächsten **Zeitschritt** (auch eng. step) auszuführenden Aktion verwendet wird. Ziel ist es, diese policy im Verlauf des Trainingsprozesses so anzupassen, dass geeignete Aktionen zur Erfüllung der Aufgabe gewählt werden. Dieser Prozess wird fortwährend wiederholt, bis der Agent in der Lage ist, das gewünschte Ziel zu erreichen oder andere terminierende Bedingungen erfüllt werden.

Um die Aufgabe zu bewältigen, ist der Agent gezwungen über mehrere Versuche Erfahrungen in der Umgebung zu sammeln. Diese durch den Agenten durchgeführten Versuche werden auch

Episode genannt und bestehen jeweils aus einer Menge an Zeitschritten. Das Ende dieser Episode kann verschiedene Ursachen haben. So können die Erfüllung der gestellten Aufgabe, neben dem Überschreiten einer maximalen Anzahl an Zeitschritten oder dem Eintreten einer terminierenden Bedingung, mögliche Gründe für ein Episodenende sein. Die konkrete Ausgestaltung der erwähnten terminierenden Bedingungen unterscheidet sich je nach Aufgabe und Umgebung, kann aber allgemein als Fehlerzustand definiert werden. Dieser macht das Erreichen des Ziels für den Agenten unmöglich oder ist als inakzeptabel im Sinne der Aufgabe zu betrachten. Neben Kollisionen mit Objekten oder dem Verlassen der Fahrspur können dies auch Zustände sein, in welchen der Agent durch blockierende Objekte bewegungsunfähig wird. Wird das Ende einer Episode erreicht, löst dies in jedem Fall ein Zurücksetzen des Agenten in den Ausgangszustand aus. So kann anschließend eine neue Episode beginnen [29, S. 6].

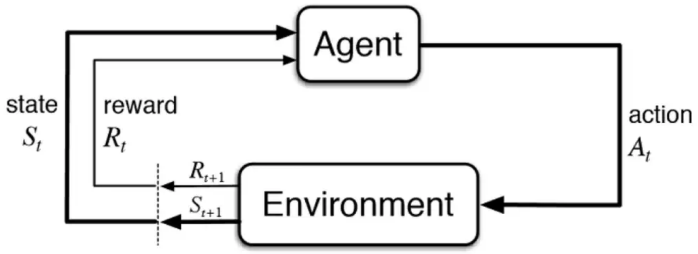


Abbildung 4.1: Interaktion Agent und Umgebung[30]

Wie in Abbildung 4.1 zu sehen, ist die Zustandsinformation s_t, auch state (dt. Status) genannt, der Beginn des Trainingsprozesses und gibt dem Agenten Auskunft über Bedingungen sowie Einflussfaktoren in seiner Umgebung, welche zum Zeitpunkt t vorliegen [31, S. 3]. Im Bereich der autonomen Fahrzeugsteuerung können dies insbesondere Informationen über die Distanz zur Mitte der Fahrbahn, die eigene Geschwindigkeit sowie (falls vorhanden) Positionen und Geschwindigkeiten anderer Verkehrsteilnehmer:innen sein [29, S. 6f]. Die Form der Zustandsinformationen variiert allerdings je nach Agent und Umgebung. So ist die Verwendung von Skalaren eine einfache Möglichkeit, im Verlauf der Simulation Parameter wie die Geschwindigkeit zu repräsentieren und an den Agenten weiterzugeben. Allerdings ist ebenso die Verwendung komplexerer Sensordaten möglich. Dies können Bilder von einer in Fahrtrichtung gerichteten Kamera oder auch Informationen eines Lidar- oder Radarsensors sein. Da es sich in diesen Fällen um unstrukturierte Daten handelt, welche erfordern, dass zunächst die relevanten Informationen (beispielsweise Fahrbahnmarkierungen) identifiziert werden, kann zur Steigerung der Effizienz des Lernprozess eine Komprimierung oder Vereinfachung dieser Daten vorgenommen werden [29, S. 6f]. Miralles [32], hat hierfür einen mehrstufigen Komprimierungsprozess durchgeführt, um unter anderem die Pixelanzahl des Bildes sowie die Komplexität des Farbraums von RGB auf Graustufen zu reduzieren. Eine mögliche Alternative zu der Komprimierung ist die Vereinfachung der unstrukturierten Daten. Hierfür kann eine semantische Segmentierung vorgenommen werden (Abbildung 4.2), welche unwesentliche Informationen aus dem Bild entfernt und eine farbliche Kontrastierung der Umgebung des Agenten vornimmt. So ist in der Abbildung deutlich zu erkennen, dass Gräser und Bäume in Grüntönen, andere Autos in blau und große Teile des Horizontes sowie der nicht zum Straßenverkehr gehörende Umgebungen durch einheitliche Farben repräsentiert werden. Ein solches Vorgehen bietet nach Aradi [29, S. 7] den Vorteil, dass Unterschiede zwischen Simulation und Realität reduziert werden können und eine anschließende Übertragung der Anwendung in reale Szenarien leichter möglich ist.

Der Agent nutzt nun die eventuell vorverarbeiteten Zustandsinformationen und reicht diese weiter an eine policy. Diese kann grundlegend als eine Art Wahrscheinlichkeitsverteilung in-

Abbildung 4.2: Semantische Segmentierung im realen Straßenverkehr[29]

terpretiert werden, welche Zustandsinformationen den Aktionen des Aktionsraums A zuordnet [31, S. 3]. Die konkrete Form der policy kann allerdings variieren. So versuchen beispielsweise Algorithmen aus dem Bereich der Wertefunktionen, den Wert beziehungsweise die Belohnung vorherzusagen, welche erreicht werden kann, wenn ausgehend vom Zustand s_t die aktuelle policy befolgt wird. Hierfür werden die Funktionen state-value (dt. Zustands-Wert) und sate-action-value (dt. Zustand-Aktion-Wert) verwendet. Der Funktionsverlauf dieser beiden Funktionen wird im Verlauf des Trainings nach und nach angepasst, sodass möglichst die Qualität der gewählten Aktionen immer weiter gesteigert werden kann. Ziel ist es, am Ende des Trainings mittels dieser beiden Funktionen im Zustand s_t, die Aktion zu ermitteln, welche die größte Belohnung verspricht [31, S. 4].

Algorithmen aus dem Bereich der policy-Optimierung hingegen verzichten auf dieses Wertefunktionen-Paar. So kann beispielsweise hier lediglich eine Gauß-Verteilung als policy verwendet werden, welche durch eine Anpassung der Parameter Standardabweichung und Mittelwert optimiert wird. Die Bestimmung der Aktion erfolgt anschließend direkt über die so parametrierte Funktion. Die Parameter der Verteilung können dann beispielsweise durch ein Gradientenverfahren im Verlauf der Trainingsiterationen angepasst werden [31, S. 5]. Methoden aus dem Bereich des **deep reinforcement learning** (DRL, dt. vertiefendes verstärkendes Lernen) hingegen verwenden für die Abbildung der policy neuronale Netze, welche aus dem Bereich **deep learning** (dt. vertiefendes Lernen) stammen. Diese werden im nachfolgenden Abschnitt genauer betrachtet und bilden die Grundlage für die im weiteren Verlauf verwendeten Algorithmen.

Unabhängig von ihrer konkreten Form wählt die policy zu jedem Zeitschritt t eine spezifische Aktion $a \in A$ aus und reicht diese anschließend an den Agenten und die Umgebung weiter. Dies überführt die Umgebung und den Agenten in den Zustand s_{t+1} [31, S. 3]. Erste reinforcement learning-Ansätze unterstützten allerdings lediglich diskrete Mengen für den zur Verfügung stehenden Aktionsraum. So eigneten sich die anfänglichen Verfahren nur für die Handhabung von Aktionsräumen, wie $A = \{a_{links}, a_{rechts}, a_{\text{vorwärts}}, a_{\text{rückwärts}}\}$. Um trotz dieser Limitierung die kontinuierlichen Wertebereiche, etwa eines Lenkwinkels oder der Gaspedalstellung abzubilden, wurde eine Transformierung auf eine diskrete Menge mit drei bis neun repräsentativen Elementen für die Abbildung des gesamten kontinuierlichen Bereiches verwendet [29, S. 2]. Bei einer solchen Transformation kontinuierlicher Mengen entstehen jedoch Unterschiede zur Realität, welche umso größer werden, je kleiner der Zustandsraum für die Transformation ist. Weitere Beispiele für diese kontinuierlichen Aktionsräume sind die Positionierung eines Roboterarms in einem Koordinatensystem sowie die Steuerung von Flugzeugen hinsichtlich ihre Höhe und Geschwindigkeit [29, S. 6].

Wurden die Zustandsinformationen durch die policy verarbeitet und eine Aktion ausgewählt, wird diese wie bereits erläutert, durch den Agenten an die Simulationsumgebung weitergeleitet. In der Simulationsumgebung erfolgt die Überführung des Agenten in den Zustand s_{t+1} sowie die Bestimmung der Belohnung für den Agenten. Die neue Zustandsinformation s_{t+1} sowie die Belohnung werden an den Agenten übermittelt. Die Höhe der erhaltenen Belohnung gibt dem Agenten darüber Auskunft, wie gut oder auch schlecht seine Aktion war. Mittels dieser Information erfolgt die Anpassung der policy, sodass in nachfolgenden Versuchen die Aufgabe erfüllt werden kann und zu jedem Zeitschritt t die erhaltene Belohnung maximiert wird [29, S. 6]. Eine policy wird optimal, wenn die summierte Belohnung über alle Zeitschritte einer Episode das Maximum erreicht [27, S. 2f]. Allerdings gibt es neben der soeben dargestellten Anpassung der policy zu jedem Zeitschritt ebenso die Möglichkeit, dies nach dem Ende einer Episode in der Häufigkeit der in dieser enthaltenen Zeitschritte durchzuführen. Diese zwei Möglichkeiten bieten verschiedene Vor- sowie Nachteile. So kann die Anwendung der Belohnung nach jedem Zeitschritt den Lernprozess beschleunigen. In einigen Fällen war allerdings das Erlernen der optimalen policy hierdurch nicht möglich. Hingegen verlangsamt die Anwendung der Belohnung nach dem Beenden einer Episode die Steigerung der Belohnung. Allerdings wird das Erreichen einer hochwertigeren policy wahrscheinlicher [29, S. 6]. Hier gilt es sorgfältig zwischen diesen Varianten abzuwägen und eine zielorientierte Auswahl der geeignetsten Lösung durchzuführen.

Im Verlauf einer Episode verfolgt der Agent die **trial and error** (dt. Versuch und Fehlschlag)-Strategie, um die Umgebung zu erkunden und ein Verständnis dafür zu entwickeln, welche Aktionen große Belohnungen versprechen. Um sein Verständnis der Umgebung auszubauen, muss der Agent zum einen neues Verhalten beziehungsweise neue Aktionen ausprobieren, aber auch auf das bereits vorhandene Verständnis der Umgebung sowie des Belohnungssystems zurückgreifen. Diese zwei Komponenten werden auch **exploration** (dt. Erkundung) und **exploitation** (dt. Verwenden) genannt und müssen sorgfältig ausbalanciert werden. Grund hierfür ist, dass jede neu erkundete Entscheidung ein Risiko birgt, da die zu erwartende Belohnung im Vorfeld nicht bekannt ist. Unter Umständen übersteigt diese Belohnung aber diejenige, welche durch ein Verhalten nach exploitation erzielt werden kann. Würde beispielsweise der Agent zu viele Aktionen nach exploration treffen, so hätte das bereits gesammelte Wissen eine zu geringe Bedeutung und würde während des Trainings selten oder auch gar nicht angewendet werden. Gegenteilig würden zu viele Aktionen nach exploitation das Erkunden der Umgebung nur in geringem Maß ermöglichen. Die Ausbalancierung dieser beiden Komponenten stellt bereits eine der größten Herausforderungen des reinforcement learnings dar [27, S. 3]. Implementierungen für die Erkundung der Umgebung können beispielsweise eine gelegentliche, rein zufallsgesteuerte Aktionswahl sein. Hierfür ist ein geeigneter Wert für $\epsilon \in [0; 1]$ zu wählen, um die Wahrscheinlichkeit der Ausführung einer zufälligen Aktion festzulegen [31, S. 9]. Ebenso kann der durch die policy gewählten Aktion zu jedem Zeitschritt ein Rauschen hinzugefügt werden. Dieses wird beispielsweise durch die Wahl eines zufälligen Wertes aus einer Gauß-Verteilung bestimmt. Die Parametrierung der Kurve beeinflusst die Größe des Rauschwertes und damit die Abweichung von der durch die policy bestimmten Aktion [33, S. 7]. Eine ausgewogene Balancierung ist in jedem Fall erforderlich und trägt maßgeblich dazu bei, dass eine geeignete policy erlangt wird [27, S. 3].

Der dargestellte Trainingsprozess zeigt, dass reinforcement learning durch die Interaktion mit der Umgebung lernt und die hierfür benötigte Datengrundlage durch diese Interaktion erzeugt. Damit unterscheidet es sich von anderen Bereichen des machine learnings, welche zunächst entsprechende Daten benötigen, bevor der Trainingsprozess beginnen kann. So verwenden Algorithmen aus den Bereichen supervised (dt. überwachtes) und unsupervised learning (dt. nicht überwachtes Lernen) große Datenmengen, welche im Vorfeld aufzubereiten und zu analysieren sind. Supervised learning verwendet neben den Daten, welche später verarbeitetet werden sollen, zusätzliche Label für den Trainingsprozess. Diese Label nehmen beispielsweise eine Klassifizie-

rung des Datensatzes oder auch eine Bewertung mittels eines Skalars vor. Ziel ist es, durch diese Label die KI mit möglichen Datenkonstellationen und den dazugehörigen gewünschten Reaktionen vertraut zu machen, damit diese im Anschluss in ähnlichen Situationen eigenständig nach aus den Daten bekannten Mustern handeln kann [27, S. 2].

Bezug nehmend auf die Problemstellung der autonomen Fahrzeugsteuerung würde ein Vorgehen nach supervised oder unsupervised learning implizieren, dass nahezu alle möglichen Zustände des Fahrzeuges sowie Zustandskombinationen mit umgebenden Fahrzeugen, Hindernissen sowie Fußgänger:innen in den Daten repräsentiert sein müssten. Bei der Simulation simpler Szenarien mit lediglich einem Fahrzeug, keinen Hindernissen sowie einer bekannten Umgebung wäre dies natürlich mit vergleichsweise geringem Aufwand möglich. Unter Einbezug von mehreren Fahrzeugen sowie beweglichen Hindernissen wird die Abbildung möglicher Kollisionen und Reaktion in den Trainingsdaten zunehmend komplex und wäre nach Ashwin und Raj [27] ein sehr zeitintensives und kostspieliges Unterfangen. Trotz dieser Vorteile gegenüber anderer machine learning Methoden sind auch Nachteile für reinforcement learning festzustellen. So ist die Abbildung realer Gegebenheiten beziehungsweise die Übertragung von Ergebnissen aus der Simulation in ein reales Umfeld eine zentrale Herausforderung. Allerdings ist es nicht immer zielführend, alle Details realer Verhaltensweisen in der Simulation abzubilden. Hier ist ein gewisser Kompromiss einzugehen, da mit der Steigerung des Detailgrades ebenso die Ausführungsdauer eines Zeitschrittes und damit auch des gesamten Trainingsprozesses steigt. Der Einfluss der Ausführungsdauer eines Zeitschrittes ist erheblich, da häufig große Mengen an Zeitschritten benötigt werden, um eine geeignete policy zu entwickeln [29, S. 4].

4.1.1 Off- und on-policy reinforcement learning

Reinforcement learning-Algorithmen sind nach unterschiedlichen Gesichtspunkten klassifizierbar. Neben den bereits dargestellten Klassen der Wertefunktionen und policy-Optimierungsverfahren sind auch weitere Klassifizierungen möglich. So ist eine Unterscheidung in off- und on-policy-Verfahren eine übergeordnetere Möglichkeit, verschiedene reinforcement learning-Methoden zusammenzufassen. Dabei wird der Aufbau der policy genauer betrachtet und eine Aufteilung in zwei Bestandteile vorgesehen. Diese werden **behavior** (dt. Verhalten) und **target** (dt. Ziel) genannt und repräsentieren zwei zentrale Aspekte einer policy. Die behavior policy ist dafür zuständig, mit der Umgebung zu interagieren, Aktionen auszuwählen und dadurch Trainingsdaten zu generieren. Die target policy hingegen soll durch den Agenten gelernt werden und wird mit den gesammelten Daten der behavior policy trainiert.

Eine Klassifizierung nach **on-policy** (dt. durch das Regelwerk) und **off-policy** (dt. abseits des Regelwerks) betrachtet die Form der Implementierung dieser beiden Bestandteile. Algorithmen nach den Prinzipien von off-policy zeichnen sich dadurch aus, dass behavior policy und target policy durch zwei voneinander getrennte policies abgebildet werden. Zusätzlich wird ein replay buffer (dt. Wiederholungsspeicher) verwendet, welcher die erzeugten Daten (Zustandsübergänge, Aktionen, Belohnungen) über mehrere Iterationen hin speichert und für die Anpassung der target policy zur Verfügung stellt. Die Verwendung der Daten in nachfolgenden Trainingsiterationen ermöglicht einen effizienten Umgang mit den generierten Daten, löst allerdings ebenso in einigen Umgebungen instabile Verhaltensweisen des Agenten aus. Dies ist darin begründet, dass die Daten im replay buffer aus dem Verhaltensmuster einer älteren policy entstanden sind. Die Verwendung dieser Daten für den Trainingsprozess kann unter Umständen die Qualität der fortgeschritteneren policy nachfolgend negativ beeinflussen. Im Kontrast hierzu nutzen on-policy-Algorithmen eine policy um sowohl behavior als auch target abzubilden und verzichten auf den Einsatz von replay buffern, was die Effizienz im Umgang mit den gesammelten Daten reduziert [34, S. 2]. Ein Beispiel für Verfahren aus dieser Kategorie sind die bereits erwähnten

policy-Optimierungsverfahren, da diese auf die Verwendung von replay buffern verzichten und die policy direkt anpassen [35, S.2].

4.1.2 Deep reinforcement learning

Klassische reinforcement learning-Ansätze haben Probleme bei der Handhabung kontinuierlicher Aktionsräume. Daher war eine Übertragung auf Anwendungen der realen Welt lange Zeit nur eingeschränkt möglich, da diese komplexere Zustände oder Aktionsräume erfordern. Entwicklungen im Bereich des deep learning, das einen weiteren Teilbereich des machine learning darstellt, ermöglichten die Steigerung der Leistungsfähigkeit und Skalierbarkeit vorheriger reinforcement learning-Lösungen. Aufgrund dieser Fortschritte entstand der Bereich des deep reinforcement learning. Algorithmen aus diesem Bereich verwenden im Kern neuronale Netze, welche gute Ergebnisse in den Bereichen der Funktionsannäherung und dem Lernen repräsentativer Werte erzielen. Wodurch diese Steigerung ermöglicht wurde. Durch die neuronalen Netze werden eine policy sowie verschieden Wertefunktionen wie beispielsweise state-value oder action-value, angenähert. Wie bei den zuvor dargestellten stochastischen Algorithmen ist hier die Verwendung von Wertefunktionen nicht zwingend erforderlich. Die Verwendung von neuronalen Netzen ermöglichen im Bereich des reinforcement learning neben der Verwendung hochdimensionaler Aktionsräume auch die Handhabung unstrukturierter hochdimensionaler Zustandsinformationen. So kommen für die Verarbeitung von Bildern sogenannte neuronale Faltungsnetze zum Einsatz, welche direkt von den in Bildern codierten Informationen lernen können. So sind DRL-Algorithmen durch die Verwendung von neuronalen Faltungsnetzen in der Lage, eine policy anhand von Bildern, welche die Zustandsinformationen darstellen, zu trainieren [31, S. 6]. Beispiele für Algorithmen, welche DRL verwenden und kontinuierliche Aktionsräume unterstützen, sind die auch später betrachteten Algorithmen DDPG, TD3 aber auch Soft Actor Critic (SAC, dt. weicher Akteur Kritiker) [28, S. 5].

Die Anpassung der Parameter des neuronalen Netzes erfolgt in der Regel über den Einsatz des Gradientenverfahrens. Hierfür dient eine Verlustfunktion als Berechnungsgrundlage, welche die Differenz zwischen einem tatsächlichen Wert und dem durch das Netzwerk vorhergesagten Wert bestimmt. Aus dieser Verlustfunktion wird der Gradient berechnet, welcher angibt, in welchem Verhältnis die Parameter des Netzes angepasst werden müssen, um in nachfolgenden Iterationen den Verlust zu minimieren. Hierfür werden aus dem bisherigen Verlauf Beispiele gesammelt, um mit ihnen einen Funktionsverlauf bilden zu können [31, S. 6]. Der mittels dieser Funktion bestimmte Fehler kann beispielsweise die Differenz zwischen der vorhergesagten Belohnung und der tatsächlich erhaltenen Belohnung sein.

4.1.3 Deep Deterministic Policy Gradient - DDPG

Die Besonderheit von DDPG gegenüber Algorithmen wie beispielsweise Q-Learning ist die deterministische Vorhersage von Aktionen. Die Grundidee der Q-Learning-Algorithmen ist, aus allen zur Verfügung stehenden Aktionen jene auszuwählen, welche die größte Belohnung verspricht. Hierfür wird aus dem aktuellem Zustand s_t für alle Aktionen ein sogenannter Q-Wert berechnet, welcher die Qualität der Aktionen beurteilt. Dies ist für diskrete Aktionsräume trivial, wird aber aufgrund der Berechnung zu jedem Zeitschritt, für kontinuierliche Räume sehr zeitaufwendig [28, S. 3].

Daher ist DDPG bei der Verwendung kontinuierlicher Aktionsräume ein beliebter Algorithmus. So ist DDPG ebenso in der Lage, Eingaben, welche hochdimensional strukturiert sind, zu verarbeiten und besitzt gute Eigenschaften bezüglich der Konvergenz der erreichten Belohnung. Grundlegend vereint DDPG Ansätze aus den Bereichen der Wertefunktionen sowie policybasierten Algorithmen und zählt durch diese Kombination zu den **actor** (dt. Akteur) **critic** (dt.

Kritiker) Algorithmen. In dieser Klasse ist es Aufgabe des actors, die auszuführenden Aktionen zu bestimmen. Hierfür verwendet er die Grundlagen der policybasierten Algorithmen und versucht im Verlauf des Trainings eine optimale policy zu entwickeln. Im Gegensatz dazu stammen die Grundsätze der Funktionsweise des critics aus dem Bereich der Wertefunktionen. Der critic bewertet zu jedem Zeitschritt die ausgeführten Aktionen und berechnet hierfür einen Q-Wert, was Ähnlichkeiten zu Q-Learning zeigt [28, S. 2]. Actor und critic werden auch **online Netzwerke** genannt und verfügen zusätzlich jeweils über ein gleichnamiges **target-Netzwerk**, welches eingesetzt wird, um den Trainingsprozess zu stabilisieren.

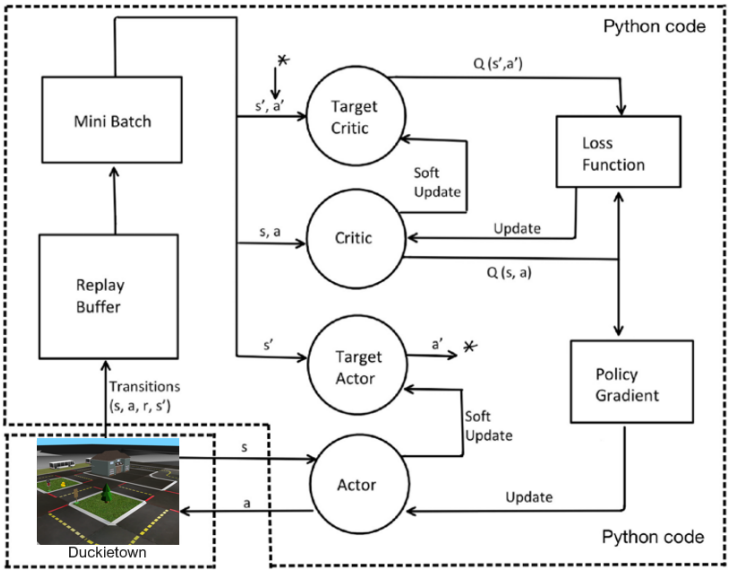


Abbildung 4.3: Schematische Darstellung DDPG, leicht verändert nach[27]

Der Ablauf des Trainingsprozesses ist in Abbildung 4.3 dargestellt. Wie abgebildet, besteht der Austausch des actors und der Duckietown-Umgebung aus den Zustandsinformationen s sowie einer gewählten Aktion a um die Umgebung in den Folgezustand s' zu versetzen. Diese Informationen werden inklusive der erhaltenen Belohnung r als Tupel in einem replay buffer gespeichert [27, S. 2f]. Einige Arbeiten führen ergänzend die Speicherung von d auf, welche Aufschluss darüber geben, ob s' terminal ist oder nicht [28, S. 3]. Allerdings ist die Größe dieses Speichers mit Bedacht zu wählen. So kann die simple Speicherung aller Zustandsüberführungen das Lernen erschweren, wohingegen ein zu kleiner Speicher zu einer Überanpassung an die gegebenen Bedingungen führen kann [28, S. 3]. Aus diesem buffer werden zufällige Tupel ausgewählt, welche einen mini Batch bilden. Die in dem mini Batch enthaltenen Daten werden im nächsten Schritt als Grundlage für die Anpassung der online und target-Netzwerke verwendet. Zunächst bestimmt der target actor mittels s' eine Aktion a', welche auf den gegebenen Zustand folgen sollte. Anschließend werden a' und s' für die Berechnung eines Ziel Q-Wertes $Q_\phi(s', a')$ an den **target critic** weitergeleitet [27, S. 2f]. Hierfür wird die nachfolgende Bellman Gleichung verwendet [28, S. 3].

$$Q_{target} = r + \gamma(1 - d)maxQ_\phi(s', a') \tag{4.1}$$

Der discount (dt. Rabatt) wird hier dargestellt durch $\gamma \in [0; 1]$ und ist bekannt aus den Grundlagen des reinforcement learnings. Wird hierfür ein kleiner Wert gewählt, verschiebt dies den Fokus auf den schnellen Erhalt einer Belohnung [31, S. 2]. Der berechnete Wert Q_{target} wird anschließend zur Konstruktion der Verlustfunktion verwendet. Hierfür berechnet der online critic zunächst einen weiteren Q-Wert $Q_\phi(s, a)$. Hierfür wird die mean squared Bellman error

(dt. mittlerer quadratischer Bellman Fehler)-Funktion verwendet. Diese bestimmt den Verlust aus dem Quadrat der Differenz des online critic Q-Wertes $Q_\phi(s,a)$ sowie Q_{target} wie auch in nachfolgender Gleichung zu sehen.

$$L = [Q_\phi(s,a) - (r + \gamma(1-d)maxQ_\phi(s',a'))]^2 \tag{4.2}$$

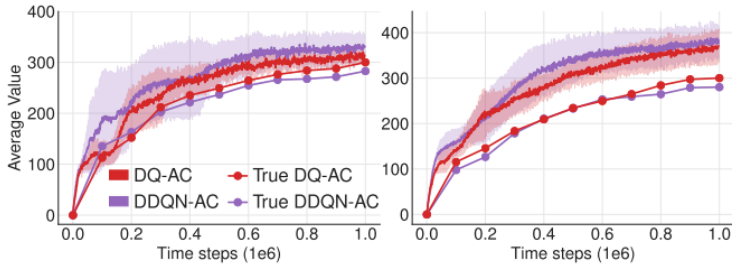
Mittels dieser Verlustfunktion erfolgt nun das Update des critics und actors mittels Gradientenabstieg, um den Verlust bezüglich der critic-Netzwerke zu minimieren. Der actor hingegen verwendet für die Maximierung des erhaltenen Q-Wertes den Gradientenanstieg [28, S. 3f]. Die Anpassung der Gewichte von target-Netzwerken erfolgt allerdings nicht über die Verwendung eines Gradienten oder der Verlustfunktion. Stattdessen erfolgt eine langsame Übertragung (auch soft update, dt. weiches Update) der Gewichte von den online-Netzwerken auf die der target-Netzwerke. Hierfür ist ein kleiner Faktor tau nahe null zu wählen, welcher bei der Übertragung von Gewichten angewendet wird. Dieser Zyklus wird nun wiederholt, indem der Agent erneut eine entsprechende Aktion auswählt [27, S. 2f].

Allerdings sind neben den Eingangs beschriebenen Vorteilen ebenso einige Nachteile von DDPG zu benennen. So reagiert DDPG sehr sensibel auf die Anpassung von Parametern wie beispielsweise der Lernraten oder dem discount. Kleine Anpassungen dieser Werte können zum Konvergieren bei einer geringen Belohnungsmenge führen oder das Netzwerk vollständig vom Lernen abhalten [28, S. 2]. Weiterhin neigen die critic-Netzwerke dazu, dass der geschätzte Q-Wert zu hoch ist. Dies kann zu einer schlechten Erkundung der Umgebung führen und das Konvergieren der erlangten Belohnung verlangsamen [28, S. 18]. Das soeben beschriebene Problem des **overestimation bias** (dt. Überschätzungsverzerrung) wird beispielsweise durch den TD3 Algorithmus adressiert, welcher auch im nachfolgenden Abschnitt betrachtet wird.

4.1.4 Twin Delayed DDPG - TD3

TD3 stellt eine Erweiterung des bereits dargestellten DDPG Algorithmus dar und versucht unter anderem das Problem des overestimation bias zu reduzieren. Dieses liegt sowohl in actor critic als auch in auf Wertefunktionen basierenden Verfahren vor und löst zu hohe Schätzungen von Belohnungen aus. Grund für dieses Verhalten ist ein kleiner Fehler, welcher bei den Schätzungen durch die critic-Netzwerke beziehungsweise Funktionen auftritt. Auch wenn dieser Fehler zu Beginn klein ist, so können sich durch ein Aufaddieren des Fehlers größere Neigungen der policy oder schlechte Updates der Netzwerke entwickeln. In Abbildung 4.4 kann dieses Problem in den Umgebungen Hopper-v1 und Walker-2d-v1 betrachtet werden. Zu sehen sind hier zum einen, die geschätzten Werte zwei verschiedener reinforcement learning-Algorithmen in Form der durchgezogenen Linie. Zum anderen wird durch die gepunktete Linie der jeweilige Optimalwert dargestellt, welcher im Optimalfall durch die Algorithmen ermittelt werden soll. Wie eindeutig zu sehen ist, liegen die Schätzungen in fast jedem Datenpunkt über den Optimalwerten. Dies zeigt, dass das Problem des overestimation bias nicht nur theoretischer Natur ist und auch in der Praxis auftaucht [33, S. 3f].

Die grundlegende Struktur von DDPG bleibt auch bei TD3 erhalten. So sind hier ebenfalls online actor und critic sowie entsprechende target-Netzwerke vorhanden. Allerdings verwendet TD3 zur Reduzierung des overestimation bias je zwei statt einem critic für die online und auch target-Netzwerke. Die Funktion der critics entspricht dabei der gleichen wie bei DDPG. Allerdings wird für die Berechnung der Verlustfunktion lediglich das Minimum der durch die beiden target-Netzwerke bestimmten Q-Werte verwendet. Dieses Vorgehen verringert das Problem des overestimation bias, welcher durch die target-Netzwerke induziert wird. Zu erwähnen ist, dass hierdurch ebenso eine Unterschätzung einiger Zustände auftreten kann, da der Q-Wert nun zu klein gewählt sein könnte. Dieser überträgt sich allerdings nicht in die policy-Updates, weshalb dieses Verhalten deutlich dem overestimation bias vorzuziehen ist [33, S. 4]. Weiterhin stellten



(a) Hopper-v1 (b) Walker2d-v1

Abbildung 4.4: Beispiel overestimation bias[33]

Fujimoto, van Hoof und Meger [33] fest, dass ein Update des actors zu jedem Zeitschritt den Fehler steigert. Dahingegen reduzieren Updates der target-Netzwerke über mehrere Zeitschritte den Fehler. Diese Erkenntnis führte dazu, dass die Autor:innen bei TD3 eine Aktualisierung des actors jeweils nach d Zeitschritten durchführen. Das bereits von DDPG verwendete Prinzip der soft Updates von target-Netzwerken findet auch bei TD3 entsprechende Anwendung. Hier ist auch die letzte Änderung von TD3 gegenüber DDPG zu finden, welche ein zufälliges Rauschen der durch den target actor bestimmten Aktion addiert. Eine Überanpassung an hohe Q-Werte kann dadurch vermieden werden. Die Implementierung dieses Rauschens ist in der nachfolgenden Berechnung des target-Q-Wertes zu sehen [33, S. 6].

$$\begin{aligned} Q_{target} &= r + \gamma(1-d)maxQ_{\phi}(s', a' + \epsilon), \\ \epsilon &\sim clip(N(0,\sigma), -c, c) \end{aligned} \tag{4.3}$$

Die Realisierung des Rauschens kann auf ähnliche Weise erfolgen wie bereits bei der Erläuterung des exploration exploitation-Dilemmas. So ist beispielsweise die Auswahl zufälliger Werte aus einer Gauß Verteilung möglich. Die so erhaltenen Werte sollten allerdings auf einen kleinen Bereich begrenzt werden. So wird beispielsweise in der Arbeit von Fujimoto, van Hoof und Meger [33] $\sigma = 0,2$ und $c = 0,5$ gewählt.

4.2 Die Simulationsumgebung gym-Duckietown

4.2.1 Karten der Simulationsumgebung

Wie zu Beginn dieser Arbeit bereits erläutert, ist gym-Duckietown eine in Python entwickelte Simulationsumgebung für die Erprobung und Erstellung von Lösungen zur Steuerung von Duckietown-Fahrzeugen. Hierfür werden im Standardumfang bereits zwölf Karten zur Verfügung gestellt, welche Situationen in unterschiedlichen Komplexitätsstufen abbilden. Als Beispiele können hier große und kleine Kreisverkehre ohne Hindernisse, aber auch komplexere Situationen aus Kreuzungsbereichen mit Hindernissen und auch Verkehrsschildern erwähnt werden. Allerdings ist es ebenso möglich, eigene Karten unterschiedlicher Komplexitätsgrade mittels einer YAML-Datei sowie den vorgefertigten Texturen zu erstellen [16].

Für die nachfolgenden Betrachtungen werden eine Auswahl der im Standardumfang enthaltenen Karten sowie die im Rahmen dieser Arbeit erstellte Karte loop empty simple (dt. Kreis leer einfach) verwendet (Abbildung 4.5). Die verschiedenen, mittels DDPG sowie TD3 erstellten policies, werden auf den Karten loop empty (dt. Kreis leer) sowie loop empty simple trainiert. Die so trainierten policies werden anschließend auf den verbleibenden Karten 4way und zig zag evaluiert. Im Vergleich zu den Trainingskarten werden hier die Grünflächen durch asphaltierte

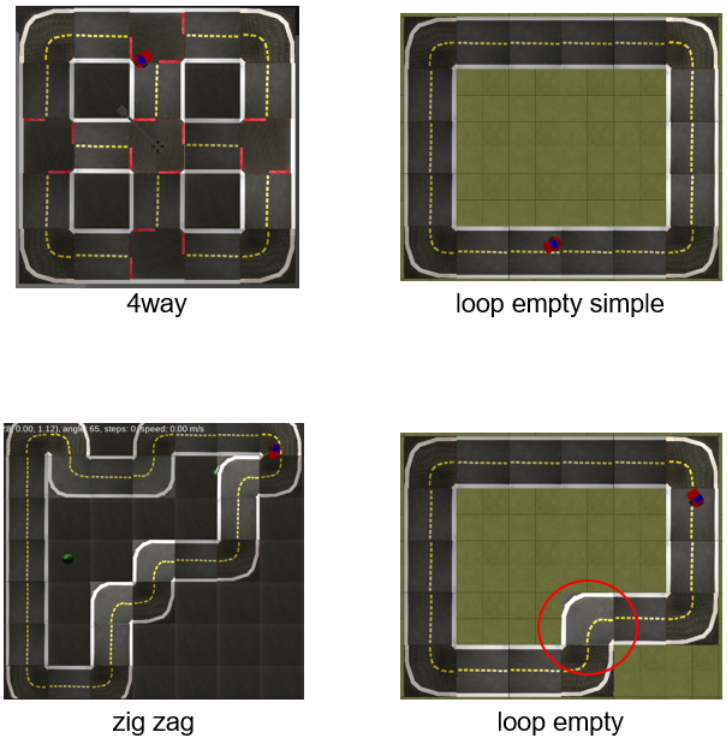


Abbildung 4.5: Karten von Duckietown [eigene Darstellung]

Bereiche ersetzt. So kann die Anpassung der policies auf ein leicht verändertes Umfeld hin überprüft werden. Durch die exemplarisch in rot markierte Kurvenkombination, stellen loop empty sowie zig zag komplexere Streckenführungen dar. Auf diesen Karten stellt sich die Frage, wie die Leistung des Agenten während und nach des Trainings auf diesen Strecken sein wird.

4.2.2 Rahmenbedingungen und Schnittstellen

Durch gym-Duckietown wird die Entwicklung verschiedener Lösungen zur Steuerung der Duckiebot-Fahrzeuge ermöglicht. So können einfachere Roboter Steuerungen, aber auch Algorithmen aus Bereichen des machine learnings realisiert werden. Dies betrifft zum einen imitation learning (dt. nachahmendes Lernen) und die hier betrachteten reinforcement learning-Algorithmen [16]. Um das Fahrzeug in der Simulationsumgebung zu steuern und die Informationen aus der Umgebung entsprechend zu verarbeiten, wird das Framework gym von OpenAI standardmäßig durch die Simulationsumgebung unterstützt. Da die Entwicklung dieses Frameworks eingestellt wurde, wird für die Aktualität der vorliegenden Arbeit der Nachfolger **gymnasium** verwendet [36]. Dieses liegt zum Zeitpunkt der Erstellung in Version 1.1.1 vor. Für die Verarbeitung und Anpassung von Informationen aus der Simulationsumgebung stellt gymnasium eine Reihe sogenannter **wrapper** (dt. Umhüllungen) zur Verfügung. Diese werden auf das Objekt der Simulationsumgebung angewendet und können anschließend Einfluss auf zurückgegebene Daten wie Zustandsinformationen, Aktionen und Belohnung nehmen. Der verwendete wrapper-Typ bestimmt hierbei, welche Informationen verändert werden. Die Verkettung verschiedener wrapper ist ebenso möglich, um eine stufenweise Anpassung der Daten vorzunehmen [37]. In Listing 4.1 ist dies anhand der Reduzierung der Bildgröße und der Umwandlung in Graustufen zu sehen.

Listing 4.1: Beispiel Funktionen gymnasium

```
1         env=Simulator()
2         wrappedEnviornment=wrappers.ResizeObservation(env,(60,80))
3         wrappedEnviornment=wrappers.GrayscaleObservation(env, True)
4
5         obs, reward, done, EvalInfo=env.step(action)
6         env.reset()
7
```

Ebenso ist mittels gymnasium die Steuerung der Umgebung sowie des Fahrzeuges möglich. So sind in den Zeilen fünf und sechs in Listing 4.1 zwei zentrale Funktionen des Frameworks zu sehen. Mittels der *step*-Funktion wird die ausgewählte Aktion (action) zunächst an gegebenenfalls verwendete wrapper und anschließend an die Umgebung für die Ausführung weitergegeben. Die verwendeten Parameter zur Steuerung des Fahrzeuges betrachten separat die Geschwindigkeit des linken und rechten Rades. Hierbei kann für jedes Rad aus dem Aktionsraum $A = [-1; 1]$ ein entsprechender Wert gewählt werden, um das Fahrzeug in die gewünschte Richtung zu bewegen. Positive Werte aus diesem Bereich ergeben für das jeweilige Rad eine Vorwärtsbeschleunigung, wohingegen negative Werte entsprechend rückwärts gerichtete Bewegungen auslösen. Allerdings ist das Fahrzeug bei einer Fahrt mit maximaler Geschwindigkeit nicht mehr in der Lage einzulenken. Aus diesem Grund wurde die maximale Geschwindigkeit auf 0,8 beziehungsweise minus 0,8 reduziert [16]. Die Rückgabewerte nach Ausführung enthalten beispielsweise die erhaltene Belohnung sowie die neue Zustandsinformation. Als Zustandsinformation wird das Bild der in Fahrtrichtung gerichteten Kamera verwendet, welche ein RGB-Bild im Format 640x480 Pixeln aufnimmt. Dieses kann ebenfalls durch den Einsatz von wrappern, vor der Ausgabe verkleinert werden, um die effiziente Verwendung von Speicher zu gewährleisten.

Die *reset*-Funktion stellt ein weiteres zentrales Element von gymnasium dar, welches verwendet wird um den Agenten in der Umgebung neu zu platzieren. Hierbei erfolgt die Platzierung zufällig auf einem geraden Straßenabschnitt auf der rechten Fahrbahnseite. Das Verlassen der Straße durch den Agenten oder das Erreichen einer Schrittanzahl von 1.500 Zeitschritten lösen in der vorliegenden Arbeit einen Aufruf dieser Methode aus und beenden damit die aktuelle Episode. Dies gilt für alle nachfolgenden Betrachtungen.

4.2.3 Belohnungsfunktion

Eine Funktion für die Bestimmung der Belohnung des Agenten ist ebenso in den Standardkomponenten von gym-Duckietown zu finden. Diese Belohnung wird dem Agenten gewährt, sobald er sich innerhalb der Grenzen der rechten Fahrbahnmarkierung bewegt, und steigt an, je näher er dem rechten Fahrbahnrand kommt. Außerdem wird eine hohe Strafe von -1000 zurückgegeben, sobald die Straße verlassen wird. Mittels dieser Funktion ließen sich jedoch im Rahmen der vorliegenden Arbeit keine guten Ergebnisse erzielen. Dieses Problem trat ebenfalls in der Arbeit von Miralles [32] auf. Hinweise auf Probleme mit der im Standardumfang enthaltenen Belohnungsfunktion sind auch im gym-Duckietown Repository zu finden, in welchem empfohlen wird, die Belohnung auf Werte zwischen null und eins zu normalisieren [16]. In der vorliegenden Arbeit wurde die Belohnung an der von Li und Okhrin [38] verwendeten orientiert. Diese basiert im Kern auf dem Winkel sowie der Distanz zur Fahrbahnmitte und der gefahrenen Geschwindigkeit. Nachfolgende Formel stellt die Berechnung der Belohnung dar.

$$R_t = \begin{cases} w_o * R_O + w_v * R_V & \text{Fahrzeug bewegt sich auf der Straße} \\ -10 & \text{Fahrzeug verlässt Straße} \end{cases} \tag{4.4}$$

Das erste Element der Belohnungsfunktion ist abhängig von der Fahrzeugposition sowie dem Winkel des Fahrzeuges in Relation zur Mitte der rechten Fahrbahn. Dieses wird obig durch R_O repräsentiert. Für die Berechnung wird eine optisch nicht sichtbare Linie in der Mitte der rechten Fahrbahn verwendet. Der Abstand des Fahrzeuges zu dieser Linie wird durch die Simulationsumgebung zu jedem Zeitschritt bestimmt. Befindet sich das Fahrzeug in geringer Entfernung zu dieser Linie und nimmt einen möglichst kleinen Winkel zu dieser ein, so erhält es eine hohe Belohnung für dieses Element. Entsprechend klein ist die Belohnung wenn beispielsweise die rechte Fahrspur verlassen wird. Der Faktor w_o bestimmt, wie groß der Einfluss dieses Elementes auf die Gesamtbelohnung des Zeitschrittes ist. In der vorliegenden Arbeit wurde ein Wert von 0,5 verwendet. Um zu vermeiden, dass der Agent lediglich still steht und keinen Vorteil aus dem Fortbewegen nehmen kann, wird auch die Fahrzeuggeschwindigkeit R_V mit einem Faktor von 0,25 in die Berechnung miteinbezogen. Die Belohnungen für Geschwindigkeit und Orientierung werden entsprechend addiert und ergeben die Belohnung für das Fahren auf der Straße je Zeitschritt. Sollte der Agent die Fahrbahn verlassen, so wird die Belohnung auf minus zehn reduziert. Diese Belohnungsfunktion wird sowohl bei den Betrachtungen bezüglich Stable Baselines 3 als auch in der eigens entwickelten Lösung im Rahmen dieser Arbeit angewendet.

4.3 Implementierung von DDPG und TD3

In den vorherigen Abschnitten wurden grundlegende Aspekte dargestellt, welche sowohl für die Betrachtungen der Stable Baselines 3 Implementierung als auch für die im Rahmen dieser Arbeit entwickelte Lösung von Relevanz sind. Der folgende Abschnitt widmet sich nun spezifisch den konzeptionellen und technischen Grundlagen der eigens implementierten Lösung. Dabei werden der Einsatz der im Framework gymnasium verwendeten wrapper, der Aufbau der zugrunde liegenden neuronalen Netze sowie die gewählte Trainingsmethodik beschrieben. Im Abschnitt 4.4 wird anschließend das Vorgehen bezüglich des Trainings mittels Stable Baslines 3 beleuchtet.

4.3.1 Implementierung der Wrapper

Mittels der im Framework gymnasium enthaltenen wrapper erfolgt während des nachfolgend dargestellten Trainingsprozesses die Anpassung von Informationen bezüglich der ausgeführten Aktionen sowie des verwendeten Kamerabildes.

Für die Reduzierung der Größe des Kamerabildes wird zunächst der observation (dt. Beobachtungs) wrapper angewendet. Dieser erlaubt eine Verkleinerung der Bildgröße von ursprünglich 640x480 auf 60x80 Pixeln. Im zweiten Schritt der Bildkomprimierung wird der Farbraum des Bildes von RGB auf Graustufen verkleinert, was den Speicherbedarf bei der anschließende Ablage im replay buffer weiter reduziert. Eine abschließende Normalisierung der Bildinformationen bildet den letzten Schritt der Vorverarbeitung des Kamerabildes. Hierdurch kann bereits die im Abschnitt 4.1 thematisierte Komprimierung der Bildgröße erreicht werden. Für die Anpassung der Fahrzeuggeschwindigkeit stehen bei nachfolgenden Trainingsversuchen sowie Evaluationen zwei Geschwindigkeitsstufen zur Verfügung. Der **fast mode** (dt. schneller Modus) wendet auf die auszuführende Aktion einen Faktor von 0,8 an. Da durch die policy Aktionen $-1 <= a <= 1$ bestimmt werden, wird so die maximale Geschwindigkeit, bei welcher das Fahrzeug noch in der Lage ist, Kurven zu fahren, erreicht. Der **slow mode** (dt. langsamer Modus) hingegen begrenzt die Fahrzeuggeschwindigkeit auf den Bereich zwischen null und 0,5.

4.3.2 Erstellung der neuronalen Netze

Das neuronale Netz, welches für die TD3 und DDPG policies verwendet wird, orientiert sich an der Lösung aus dem gym-duckietown Repository [16]. Für die Extraktion der Bildeigenschaften

werden vier **convolutional layer** (dt. Faltungslagen) mit einer Kernelgröße von vier und einem stride (dt. Schreiten) von zwei verwendet. Ausnahme bildet der erste layer, welcher eine Kernelgröße von acht verwendet. Nach jedem dieser Layer erfolgt eine Batch Normalisierung (dt. Stapelnormalisierung).

Bei den nachfolgenden **linearen Layern** erfolgt eine Unterscheidung zwischen actor und critic. So werden für den critic drei lineare Layer verwendet, wohingegen der actor lediglich zwei lineare Layer einsetzt. Als Aktivierungsfunktion wird in beiden Fällen eine leaky ReLU-Funktion verwendet. Um eine Aktion für das linke als auch rechte Rad zu bestimmen, sind durch das actor-Netzwerk bekanntermaßen zwei Ausgabewerte zu schätzen. Damit diese möglichst dem verwendeten Aktionsraum entsprechen, wird auf den Ausgabewert des linken Rades eine Sigmoid Funktion und auf den des rechten Rades eine hyperbolische Tangensfunktion angewendet. Die Ausgabe des critics hingegen ist ein linearer layer, welcher die Q-Werte für die jeweilige Aktion zurückgibt.

4.3.3 Methodik für das Training der Algorithmen

Grundlage des Trainingsprozesses bilden die zuvor dargestellten wrapper sowie neuronalen Netze. Die verwendeten Parameter für das Training können Tabelle 4.2 entnommen werden. Hier ist zu sehen, dass auf der Strecke loop empty das Training im bereits thematisierten slow mode durchgeführt wurde. Dies lässt sich damit begründen, dass der Agent bei Verwendung des fast modes häufig an der rechts-links Kurvenkombination scheiterte. Dieses Scheitern führte in den meisten Trainingsläufen zu einem Fahrtrichtungswechsel des Fahrzeuges bei Erreichen der kritischen Kurve, sodass der Agent versuchte die Strecke in die andere Richtung zu befahren, um anschließend erneut bei Erreichen der Kurvenkombination die Fahrtrichtung zu wechseln. In einigen Trainingsläufen neigte das Fahrzeug sogar dazu, gar keine Kurven mehr innerhalb der eigenen Fahrspur zu durchfahren und pendelte lediglich auf einer der Geraden. Eine Strafe beziehungsweise eine Reduzierung der Belohnung erfährt der Agent hierbei nur in einem geringem Maße, da nach dem Richtungswechsel erneut die rechte Fahrbahnseite befahren und dies entsprechend belohnt wurde. Eine sinnvolle Erweiterung an dieser Stelle könnte die Implementierung eines Strafterms sein, welcher etwaige Veränderungen der Fahrtrichtung sanktioniert.

Policy	**Eigens implementiert**		**Stable Baselines 3**	
	DDPG	TD3	DDPG	TD3
Lernrate actor	$1 * 10^{-4}$	$1 * 10^{-4}$	$1 * 10^{-4}$	$1 * 10^{-4}$
Lernrate critic	$1 * 10^{-4}$	$1 * 10^{-3}$	$1 * 10^{-4}$	$1 * 10^{-4}$
Aktionsrauschen	0,2	0,2	0,2	0,2
Karte : loop empty simple				
Fahrzeug-geschwindigkeit	fast mode	fast mode	fast mode	fast mode
replay buffer (steps)	500.000	500.000	500.000	500.000
Karte: loop empy				
Fahrzeug-geschwindigkeit	slow mode	slow mode	-	-
replay buffer (steps)	100.000	250.000	-	-

Tabelle 4.2: Trainingsparameter [eigene Darstellung]

Die Durchführung des Trainings erfolgte nach den oben dargestellten Grundlagen von DDPG und TD3 mittels des deep learning Frameworks Pytorch. Aus diesem Framework konnten einige Implementierungen verwendet werden, welche sich für den Einsatz bei reinforcement learning Problemen eignen. So wurde beispielsweise für die Speicherung der Zustandsübergänge die Klasse *ReplayBuffer* eingesetzt. Die Größe dieses replay buffers wurde, wie in Tabelle 4.2 zu sehen, auf der Strecke loop empty sowohl für DDPG als auch TD3 reduziert. Grund hierfür ist der bereits im Abschnitts 4.1.1 thematisierte Einfluss der replay buffer Größe, welche sich auf den Erfolg des Trainings auswirkt. Bei Trainingsdurchläufen auf dieser Strecke mit großen replay buffern konnte nach einer gewissen Anzahl an Zeitschritten ein Einbrechen der Belohnung festgestellt werden. Dieses Verhalten deutet auf einen zu starken Einfluss vergangener beziehungsweise nicht lohnenswerter Transitionen hin. Die Reduzierung der Speichergröße verringerte diesen Effekt, da ältere Zustandsübergänge schneller durch neue Erfahrungen überschrieben wurden.

Wird der Trainingsprozess gestartet, erfolgt bis zum Erreichen von 5.000 Zeitschritten die Auswahl der Aktionen nicht durch den actor. Stattdessen werden zunächst zufällig ausgewählte Aktionen ausgeführt. Wird diese Anzahl an Zeitschritten überschritten, erfolgt der bereits dargestellte Prozess unter Einbezug des actors. Mittels dieses Vorgehens konnte die Qualität der Ergebnisse schneller gesteigert werden. Für den Zeitpunkt der durchzuführenden Trainingsschritte gibt es wie bereits in Abschnitt 4.1 dargestellt, die Möglichkeit, dies nach dem Ende einer Episode oder zu jedem Zeitschritt zu tun. Hierbei stellte sich heraus, dass durch ein Update der Netzwerke nach dem Ende einer Episode in der Tat höhere Belohnungen erzielt werden konnten.

4.4 Implementierung mit Stable Baselines 3

Das Framework Stable Baselines ist eine Sammlung von reinforcment learning-Algorithmen, welche auf verschiedene, in gymnasium implementierte Umgebungen angewendet werden können. Zum Zeitpunkt der Erstellung dieser Arbeit im Frühjahr 2025 liegt Stable Baselines in der ersten Generation, welche unter Stable Baselines bekannt ist und in der aktualisierten Fassung Stable Baselines 3 vor. Unterschied dieser beiden Derivate ist unter anderem das verwendete deep learning Framework. So verwendete die erste Generation Tensorflow und unterstützt dieses bis zu der Version 1.8. Die aktuelle Version Tensorflow 2 wird nicht unterstützt. Dahingegen setzt Stable Baselines 3 das Framework Pytorch ein, welches wie bereits erwähnt, ebenso für die Ergebnisse dieser Arbeit verwendet wurde. Um Unterschiede beispielsweise in der Laufzeit oder in der allgemeinen Performance auszuschließen, wird nachfolgend Stable Baselines 3 als Vergleich verwendet. Ebenso stellt die Verwendung der aktualisierten Version einen aktuelleren Bezug der Arbeit her [39].

Die Einbindung verschiedener Simulationsumgebungen, welche durch OpenAI gym oder gymnasium unterstützt werden, ist in Stable Baselines 3 möglich. Neben den wrappern, welche bei der Initialisierung der Umgebung durch gymnasium hinzugefügt werden, ergänzt Stable Baselines 3 weitere wrapper für die **Vektorisierung** der Umgebung. Durch diese Vektorisierung ist weiterhin ein paralleles Training von Agenten in mehreren Umgebungen möglich, welches aber in der vorliegenden Arbeit nicht verwendet wurde. Die im vorherigen Abschnitt verwendeten wrapper für die Umwandlung in Graustufen sowie die Normalisierung des Bildes wurden bei der Implementierung der Stable Baselines 3 Algorithmen nicht verwendet. Grund hierfür ist die mangelnde Performance, welche bei der Verwendung von diesen wrappern festzustellen war. Ein Grund dafür kann in der Art und Weise liegen, wie die Vektorisierung die Form der Daten beeinflusst. Die Größe des Kamerabildes wird allerdings für die Umsetzung mittels Stable Baselines 3 ebenso auf 60x80 Pixeln reduziert [8].

Für das Training von Modellen sind in dem Framework bereits eine Reihe von neuronalen Netzen für die Erstellung der policy implementiert. Für DDPG als auch TD3 werden hier

grundlegend die gleichen Strukturen eingesetzt. Für die nachfolgenden Betrachtungen wurde die Verwendung der im Framework enthaltenen CNN policy sowie die Übertragung der in Abschnitt 4.3.2 dargestellten Struktur geprüft. Da mit der in Abschnitt 4.3.2 beschriebenen Implementierung keine Verbesserung der Ergebnisse erzielt werden konnte, wurde die Standardimplementierung als am sinnvollsten erachtet. Die verwendeten Parameter können entsprechend aus der Tabelle 4.2 entnommen werden.

Als Besonderheit ist darauf hinzuweisen, dass eine Konfiguration unterschiedlicher Lernraten für actor und critic in Stable Baselines 3 nicht ohne Weiteres unterstützt wird. Durch Änderungen an dem Quellcode des Frameworks, konnten jedoch Versuche mit unterschiedlichen Lernraten für actor und critic durchgeführt werden. Allerdings blieb hierbei eine Veränderung der Ergebnisqualität aus. Dieses Verhalten kontrastiert die Ergebnisse, welche bei der Implementierung ohne Stable Baselines 3 festgestellt wurden. Bei dieser war ein erfolgreiches Training ohne eine Differenzierung der Lernraten von actor und critic nicht möglich. Eine mögliche Erklärung für diese unterschiedlichen Verhaltensweisen kann die Vektorisierung der Simulationsumgebung in Stable Baselines 3 sein, welche Einfluss auf die Dimensionierung der Eingabeparameter von den policies nimmt. Eine Überprüfung dieser Theorie in der Praxis erfolgt nachfolgend nicht, da die Vektorisierung nicht ohne weiteres abgeschaltet werden kann.

Als Rauschen zur Erkundung der Umgebung wurde ebenfalls eine Gauß-Verteilung verwendet. Diese ist jedoch in Stable Baselines 3 zustandsabhängig gestaltet, was das Verhalten des Fahrzeuges in der Realität optimieren soll [8]. Bei der Implementierung dieser Arbeit werden zu Beginn des Trainings eine Reihe zufälliger Schritte durchgeführt. Dieses Vorgehen wird standardmäßig nicht von Stable Baselines 3 unterstützt. So wird lediglich die Ausführung zufälliger Aktionen zur Füllung des replay buffers ohne ein Training von actor sowie critic ermöglicht.

4.5 Evaluation der Ergebnisse

Für die Betrachtung der Ergebnisse dieser Arbeit werden zunächst die Trainingsverläufe der policies auf den Karten loop empty und loop empty simple dargestellt. Anschließend werden die Ergebnisse auf den eben genannten Karten sowie 4way und zig zag evaluiert. Ebenso wird der:die Leser:in in die verwendete Metrik zur Bewertung der Ergebnisse eingeführt. Um nachfolgend eine Unterscheidung der auf unterschiedlichen Karten trainierten policies zu erleichtern, werden zur Kennzeichnung der für das Training verwendeten Karten das Präfix LE (loop empty) und LES (loop empty simple) für den Algorithmus verwendet (Beispiel: LE-TD3). Für eine Differenzierung zwischen der Implementierung dieser Arbeit und Stable Baselines 3 wird das Suffix stb3 (Beispiel: LES-TD3-stb3) bei Betrachtungen bezüglich Stable Baselines 3 ergänzt.

4.5.1 Trainingsverlauf

Um die Entwicklung der Qualität der policy zu betrachten und potentielle Unterschiede im Trainingsverlauf zwischen den Lösungen festzustellen, wurde in regelmäßigen Abständen von zehn Episoden eine Evaluierungsphase durchgeführt. Jede dieser Phasen besteht aus insgesamt zehn Episoden, welche ebenfalls auf der jeweiligen für den Trainingsprozess verwendeten Karte durchgeführt werden. Während jeder Evaluierungsepisode wird die jeweils erreichte Gesamtbelohnung bestimmt und abschließend über alle zehn Evaluierungsepisoden gemittelt. Eine policy-Evaluierung von der im Rahmen dieser Arbeit entstandenen Lösungen wird ohne jegliche Addition eines Aktionsrauschen zur Erkundung durchgeführt.

Evaluierungen im Kontext des Stable Baselines 3 Frameworks hingegen setzen die Verwendung des Aktionsrauschens während der Evaluierung fort. Dies ist ebenso bei den später betrachteten Experimenten mit den trainierten policies der Fall. Ein Entfernen dieses Rauschen bei der nachfolgenden Evaluierung löste starke Verhaltensänderungen sowie ein Abfallen der erreichten

Belohnung aus. Grund hierfür kann eine Überanpassung der neuronalen Netze an die Präsenz dieses Rauschen sein. Eine solche Überanpassung erschwert es der policy sich auf neue Gegebenheiten wie das Fehlen des Rauschen einzustellen und unter diesen geänderten Bedingungen die gleiche oder eine ähnliche Performance zu leisten. Eine solche Überanpassung wird ebenfalls bei den nachfolgenden Betrachtungen bezüglich Stable Baselines 3 deutlich.

Eigens erstellte und implementierte Lösung

Die eigens erstellte Lösung wurde auf den Karten loop empty simple und loop empty trainiert. In Abbildung 4.6 ist zunächst die Entwicklung der mittleren Belohnung auf beiden Strecken im Rahmen der Evaluierungsepisoden zu sehen. Auffällig sind zum einen die Schwankungen, welchen die erzielte Belohnung unterliegt. So können beispielsweise im Falle von LES-DDPG auf der Strecke loop emtpy simple zu Beginn des Trainings Einbrüche der Belohnung von ca. 1300 auf lediglich 700 festgestellt werden. Ähnliche Schwankungen sind auch im Falle von TD3 zu erkennen, allerdings ist die Schwankungsbreite hier etwas geringer. Das Auftreten dieser Schwankungen kann durch ein leicht instabiles Lernverhalten erklärt werden. Als mögliche Ursache kann hier eine zu starke Anpassung der Parameter durch den Gradientenabstieg angeführt werden. Weiterhin ist die erreichte Belohnung auf dieser Strecke bis auf wenige Ausnahmen von LES-TD3 meist höher als von LES-DDPG. Da TD3 die Weiterentwicklung von DDPG ist, entspricht dies auch dem zu erwartenden Verhalten. Das Konvergieren der Belohnung bei einem Wert von etwa 1.500 ist bei LES-TD3 zudem zu einem etwas früheren Zeitpunkt festzustellen.

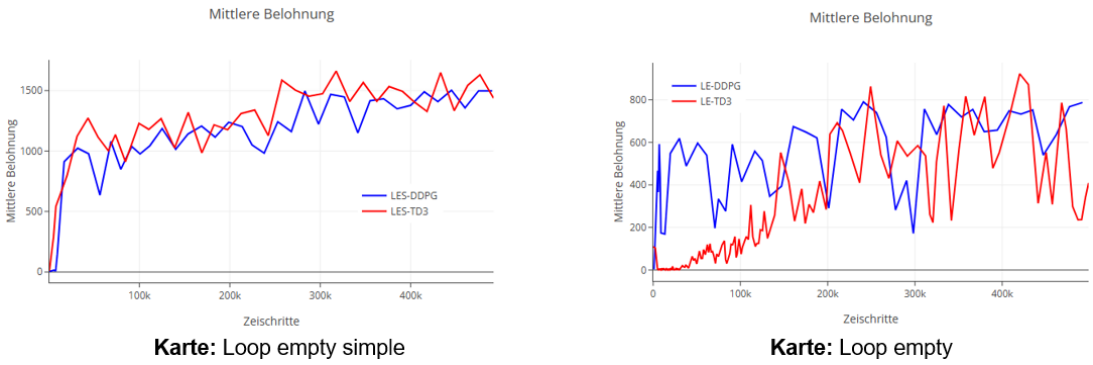


Abbildung 4.6: Verlauf Belohnung auf loop empty und loop empty simple [eigene Darstellung]

Die Graphen bezüglich der Strecke loop empty weisen ebenso einige Besonderheiten auf. Im Vergleich zu dem Verlauf des Trainings auf der Karte loop empty simple sind bei LE-TD3 die Schwankungen der Belohnung deutlich stärker. Diese führen bei den erzielten Ergebnissen zu einem Reduzieren der Belohnung kurz vor Erreichen des Trainingsendes, wodurch die Ergebnisse unter denen von LE-DDPG liegen. Solche Tendenzen zu einem starken Abfall der Belohnung konnten allerdings während des Trainings beider Algorithmen festgestellt werden. So sank in einigen Trainingsversuchen die Belohnung von ein Tausend bis auf null und stieg anschließend nur langsam wieder an. Ein solches Verhalten kann einen Zusammenhang zu einer falschen Größe des replay buffers haben. Wie in Abschnitt 4.1.1 beschrieben, können sich sowohl zu große als auch zu kleine Speicher negativ auf das Trainingsverhalten auswirken. Bestätigt werden konnte dieser Verdacht durch ein sukzessives Reduzieren der Größe des replay buffers, welches die Schwankungen auf das in Abbildung 4.6 dargestellte Niveau begrenzte.

Stable Baselines 3

Der Verlauf des Trainings auf der Karte loop empty simple mittels Stable Baselines 3 ist in Abbildung 4.7 für LES-DDPG-stb3 als auch LES-TD3-stb3 zu sehen. Der Graphenverlauf für die Belohnung unterliegt lediglich geringen Schwankungen und zeigt in beiden Fällen ein gutes konvergierendes Verhalten. Wie bereits bei der im Rahmen dieser Arbeit implementierten Version ist TD3-stb3 in der Lage, früher als DDPG-stb3 zu konvergieren und erreicht zu einem deutlich früheren Zeitpunkt hohe Belohnungen. Auffällig ist allerdings eine geringe Steigung von LES-DDPG-stb3, welche Anlass für eine Verlängerung der Trainingsdauer auf eine Millionen Zeitschritte war. So konnte die erzielte Belohnung in etwa auf das Niveau von TD3-stb3 gesteigert werden.

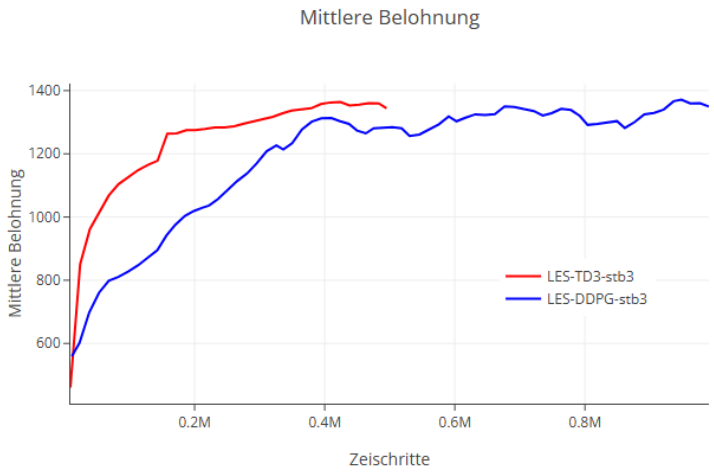


Abbildung 4.7: Verlauf Belohnung Stable Baselines 3 auf loop empty simple [eigene Darstellung]

Das dargestellte gute konvergierende Verhalten beider Algorithmen von Stable Baselines 3 stellte ebenfalls eine Herausforderung dar. So neigten unter Verwendung von gleichen Parametern die policies häufig dazu, bei einer geringen erhaltenen Belohnung zu konvergieren. Einen Lernerfolg, wie er in Abbildung 4.7 zu sehen ist, wurde erst nach einigen Trainingsläufen erreicht. Dieses stark variierende Verhalten deutet daraufhin, dass die mittels zufälliger Werte initialisierten Parameter der neuronalen Netze im Verlaufe des Trainings nicht immer im notwendigen Maß angepasst werden konnten. Dieses Problem konnte weder durch eine Anpassungen der Lernrate noch durch eine Anpassung des Aktionsrauschens behoben werden. Als Ursache hierfür kann zum einen die vorliegende Vektorisierung der Umgebung durch das Framework genannt werden, welche Einfluss auf das Lernverhalten nehmen kann. Zum anderen ist es ebenso möglich, dass während des Trainings zu große Anpassungen der policy-Parameter durch regulatorische Eingriffe seitens des Frameworks vermieden werden. Hierfür sprechen auch die geringen Schwankungen, welche im Graphenverlauf zu sehen sind.

Vergleich

Bei einer Gegenüberstellung der beschriebenen Trainingsverläufe sowie der aufgetretenen Probleme wird deutlich, dass beide Implementierungen Schwankungen aufweisen. Die Auswirkungen dieser sind allerdings unterschiedlich. In der eigens erstellten Version im Rahmen dieser Arbeit, waren Schwankungen in Form einiger Höhen und Tiefen im Verlauf eines Trainingsverlaufs festzustellen. Diese zeigten sich je nach Umgebung in verschiedener Intensität, verhindert allerdings nicht den Lernerfolg der policy. Bei Stable Baselines 3 hingegen waren Schwankungen über die Grenzen eines Trainingslaufes hinaus zu erkennen. Diese verursachten stark unterschiedliche Be-

lohnungen, welche die Qualität der policy stark reduzierten. Ein solches Verhalten ist als äußerst hinderlich anzusehen und kann den Entwicklungsprozess stark beeinflussen.

Algorithmus	Anzahl steps	steps pro Sekunde	Gesamtdauer
DDPG	500.000	35,61	3,9h
TD3	500.000	35,52	3,91h
DDPG-stb3	1 Mio.	74,47	3,73h
TD3-stb3	500.000	83,66	1,66h

Tabelle 4.3: Laufzeiten der Experimente [eigene Darstellung]

Ein weiterer interessanter Aspekt ist die benötigte Laufzeit für das Training, da hier deutliche Unterschiede zwischen Stable Baselines 3 und der eigens implementierten Lösung festzustellen sind. Die benötigten Laufzeiten sowie die durchschnittlich durchgeführten Zeitschritte je Sekunde können Tabelle 4.3 entnommen werden und zeigen einen höhere Performance seitens Stable Baselines 3. So wird bei einem Vergleich dieser Werte deutlich, dass eine Ausführung mittels Stable Baselines 3 die Laufzeit um mehr als die Hälfte reduziert. Dies wird besonders bei einer Betrachtung der ausgeführten Zeitschritte je Sekunde deutlich. Dieser Vorteil kann auf den Einsatz effizienterer Bibliotheken beziehungsweise Implementierungen oder die Vektorisierung der Umgebung zurückgeführt werden. Auch sind leichte Unterschiede der Laufzeit zwischen Algorithmen innerhalb einer Lösung festzustellen. Allerdings können diese auf die unterschiedliche Dauer der Evaluierungsphasen während des Trainings zurückgeführt werden. Diese kann beispielsweise sehr kurz ausfallen, wenn der Agent zu Beginn der Evaluierungsepisode die Straße verlässt. Fährt er hingegen wie gewünscht auf der rechten Fahrspur und verlässt damit die Straße nicht, so ist die Dauer der Evaluierung entsprechend länger, was ebenfalls die Dauer des Trainingsprozesses erhöht.

4.5.2 Bewertung der Modellergebnisse

Neben der bereits eingeführten Belohnung, bestehend aus Geschwindigkeit sowie Winkel und Position des Fahrzeuges, werden in der nachfolgenden Betrachtung weitere Metriken zur Bewertung der Ergebnisse herangezogen. So wird die mittlere Geschwindigkeit des Fahrzeuges, die mittlere Dauer einer Episode sowie die Belohnung für die Fahrzeugposition betrachtet. Hierdurch wird ein besserer Einblick in eventuelle Neigungen des Agenten ermöglicht, die Belohnung entweder durch eine hohe Geschwindigkeit oder eine gute Positionierung zu steigern. Für die Strecken 4way sowie loop empty simple werden in nachfolgenden Betrachtungen der fast mode verwendet. Der slow mode wurde auf den etwas komplexeren Strecken zig zag sowie loop empty verwendet, um das Passieren der bereits thematisierten kritischen Kurven zu erleichtern.

Ergebnisse in der Trainingsumgebung

Die Metriken, welche die verschiedenen policies in der ihnen bekannten Umgebung erreichen, können Tabelle 4.5 entnommen werden. Hier ist anhand der Daten zu erkennen, dass sowohl die Implementierung im Rahmen dieser Arbeit als auch Stable Baselines 3 in der Lage sind, unter jedem Gesichtspunkt dieser Metrik hohe Werte zu erreichen. Auffällig sind in beiden Implementierungen die meist bessere Performance von TD3 im Vergleich zu DDPG. Dies ist nicht nur bei der mittleren Belohnung zu erkennen, sondern auch bei der mittleren Geschwindigkeit. Weiterhin ist dies auch bei der Belohnung für die Fahrzeugposition der Fall. Abweichungen sind allerdings im Bereich der Lösungen mittels Stable Baselines 3 festzustellen. So erzielt hier zwar LES-TD3-stb3 ebenfalls eine höhere mittlere Belohnung als LES-DDPG-stb3, erreicht in den Bereichen mittlere Geschwindigkeit sowie mittlere Dauer einer Episode jedoch nicht die Werte

von DDPG. Dies deutet darauf hin, dass die höhere erreichte Belohnung in weiten Teilen durch eine bessere Positionierung des Fahrzeuges erzielt werden konnte. Dies stellt einen Kontrast zu der Implementierung dieser Arbeit dar, da hier mittels TD3 ebenso höhere Geschwindigkeiten erreicht werden konnten. Die Gesamtbetrachtung aller Metriken zeigt allerdings, dass die Implementierungen mittels Stable Baselines 3 im direkten Vergleich in der Lage sind, eine höhere Belohnung zu erzielen und lediglich in dem Bereich der Fahrzeuggeschwindigkeit hinter den Ergebnissen der Implementierung dieser Arbeit zurückbleiben.

Metrik	**LES-DDPG**	**LES-TD3**	**LES-DDPG-stb3**	**LES-TD3-stb3**
Mittlere Belohnung	1274.68	1348.93	1322.04	1408.86
Mittlere Geschwindigkeit	0.76 m/s	0.81 m/s	0.79 m/s	0.72 m/s
Mittlere Belohnung Positionierung	0.63	0.68	0.64	0.72
Mittlere Dauer je Episode	13.67	14.01	14.63	13.48

Tabelle 4.5: Ergebnisse in Umgebung loop empty simple [eigene Darstellung]

Bei der visuellen Bewertung des Fahrverhaltens der eigens implementierten Algorithmen ist zu erkennen, dass LES-TD3 in der Lage ist die gelernte Strecke loop empty simple stabil abzufahren. Bei der Ausführung von LES-DDPG können hingegen gelegentliche ruckartige Lenkbewegungen nach links und rechts festgestellt werden. Dies könnte eine Auswirkung des bereits erläuterten overestimation bias-Phänomenes sein, welches hier dazu führt, dass im Wechsel Positionen weiter links beziehungsweise rechts von der aktuellen Fahrzeugposition als lohnenswerter erachtet werden. Ebenso könnte dies eine Auswirkung der für das Aktionsrauschen verwendeten Gauß-Verteilung sein. So gehen Raffin, Kober und Stulp [8] in ihrer Arbeit davon aus, dass bei der Verwendung eines zustandsunabhängigen Rauschens häufig ein „shaky behavior “ (dt. Schüttelndes Verhalten) feststellbar ist. Da in der vorliegenden Lösung ein solches Rauschen verwendet wurde, kann dies eine mögliche Ursache sein. Die Implementierung eines zustandsabhängigen Rauschens, wäre für DDPG als auch für TD3 von Interesse und kann als Forschungsdesiderat betrachtet werden.

Das Fahrverhalten von LES-TD3-stb3 und LES-DDPG-stb3 auf loop empty simple ist visuell ähnlich. Beide policies sind in der Lage, den Kurs mit einer relativ hohen Geschwindigkeit abzufahren und können meist eine hohe Belohnung erzielen. Allerdings werden gelegentlich Situationen erreicht, welche zu einem unkontrollierten Verhalten oder zu einem Stillstand des Fahrzeuges führen. Der Agent ist in diesen Situationen mit der vorliegenden policy nicht in der Lage, diese Position des Stillstandes zu verlassen oder andere unkontrollierte Verhalten zu beenden. Grund hierfür kann ebenfalls das bereits thematisierte overfitting durch Stable Baselines 3 sein, welches ein instabiles Verhalten des Agenten in einigen Fahrzeugpositionen auslöst.

Die auf der Strecke loop empty trainierten policies zeigen in Tabelle 4.6 ein etwas anderes Bild. So liegt hier LE-TD3 in den Bereichen der mittleren Belohnung sowie der Belohnung für die Fahrzeugpositionierung deutlich hinter LE-DDPG zurück, was nicht zuletzt an dem im Abschnitt 4.5.1 thematisierten Verhalten liegt. Zu erkennen ist allerdings, dass hier TD3 ebenso dazu neigt, deutlich höhere Geschwindigkeiten als LE-DDPG zu verwenden. Dies stellt eine Parallele zu den Beobachtungen auf der Strecke loop empty simple dar. Eine weitere Parallele kann bei einer visuellen Beobachtung der Fahrverhalten festgestellt werden. So sind auch hier bei der Ausführung von LE-DDPG gelegentliche ruckartige Lenkbewegungen nach links und rechts festzustellen. Diese sind allerdings etwas stärker ausgeprägt als auf der Strecke loop empty simple, was auf Schwierigkeiten bei der Anpassung der policy an die Gegebenheiten der Strecke hindeuten kann.

Metrik	LE-DDPG	LE-TD3
Mittlere Belohnung	732.74	448.73
Mittlere Geschwindigkeit	0.34 m/s	0.52 m/s
Mittlere Belohnung Positionierung	0.34	0.32
Mittlere Dauer je Episode	15.20	7.30

Tabelle 4.6: Ergebnisse in Umgebung loop empty [eigene Darstellung]

4.5.3 Experimente in anderen Umgebungen

Ergänzende Vergleiche aller vorhandenen Lösungen wurden auf Strecken durchgeführt, die nicht Teil des Trainingsprozesses waren. Insbesondere sind die Strecken 4way und zig zag für alle policies vollständig unbekannt. Diese Strecken unterscheiden sich sowohl in der Streckenführung als auch in der Gestaltung der Flächen abseits der Straße von den Trainingsumgebungen. Während in den Trainingsumgebungen diese Flächen durch Rasen dargestellt wurden, weisen die Strecken 4way und zig zag asphaltierte Flächen abseits der Straße auf.

Zunächst werden die Ergebnisse auf der Strecke loop empty betrachtet. Hier sind bei den auf der Strecke loop empty simple trainierten policies verschiedenen Unterschiede festzustellen. So sind zwar die Implementierungen aus Stable Baselines 3 in dieser Umgebung in der Lage, Belohnungen von 254,4 beziehungsweise 264,09 zu erreichen, jedoch ist die mittlere Dauer jeder Episode sehr gering. Beispielsweise erreicht LES-TD3-stb3 hier lediglich einen Wert von 6,54 Sekunden. Dahingegen sind LES-DDPG sowie LES-TD3 in der Lage, eine deutlich höherere Episodendauer und teilweise auch höhere Belohnung zu erzielen. Dies ist insbesondere bei LES-TD3 zu beobachten, welches auch die einzige nicht auf dieser Strecke trainierte policiy ist, welche den Kurs vollständig bewältigen kann. LES-DDPG sowie die Stable Baselines 3 policies scheitern meist an der rechts-links Kurvenkombination dieser Strecke.

Die bereits thematisierte Überanpassung der Implementierung von Stable Baselines 3 kann bei den nachfolgenden Betrachtungen auf der Strecke 4way sowie zig zag erneut festgestellt werden. So sind die policies dieses Frameworks auf beiden Strecken nicht in der Lage, hohe Werte in einem Bereich der Metrik zu erzielen. Visuell äußert sich dies in einem schlechten Fahrverhalten, welches durch häufige und schnelle Richtungswechsel sowie Kontakte mit der Fahrbahnbegrenzung geprägt ist. Da die Strecke zig zag durch die zahlreichen Kurven um einiges komplexer ist als die für das Training verwendete Strecke loop empty simple, war hier in Teilen eine schlechtere Leistung zu erwarten. Auf der Strecke 4way hingegen war, aufgrund des ähnlichen Layouts im Vergleich zu der Karte loop empty simple von LES-DDPG-stb3 sowie LES-TD3-stb3 eine gute Performance zu erwarten. Diese Annahme wurde allerdings nicht bestätigt. Grund hierfür könnten die thematisierten Unterschiede in der Gestaltung der Umgebung abseits der Straße sein. Es ist davon auszugehen, dass die mittels Stable Baselines 3 erstellten policies, neben der Fahrbahnbegrenzung ebenso die Grünfläche für die Orientierung verwendeten. Aus diesem Grund könnten Probleme in Umgebungen auftreten, in welchen diese grünen Flächen nicht vorhanden sind.

Im Kontrast hierzu war die Implementierung dieser Arbeit in der Lage, auf der Strecke 4way eine mittlere Belohnung von über 900 zu erzielen. Ein Umfahren der Strecke konnte bis auf wenige Ausnahmen durch diese policies erreicht werden. Dies zeigt die bessere Anpassungsfähigkeit der policies an eine geänderte Umgebung. Dennoch kann anhand der mittleren Episodendauer festgestellt werden, dass auch hier die Fahrt häufig durch ein Verlassen der Straße beendet wurde.

5 Zusammenfassung und Ausblick

In der vorliegenden Arbeit wurden die reinforcement learning-Algorithmen DDPG sowie TD3 in der Simulationsumgebung Duckietown miteinander verglichen. Hierbei wurden Implementierungen mittels des Frameworks Stable Baselines 3 und einer eigens erstellten Version berücksichtigt, welche es zum Ziel hatten den Duckiebot mithilfe von Fahrspurerkennung und Objektidentifzierung durch die Simulationsumgebung zu bewegen. Stable Baselines 3 ist ein hochaktuelles und weitverbreitetes reinforcement learning Framework, welches Implementierungen verschiedener reinforcement learning Algorithmen zur Verfügung stellt. Ein relevanter Aspekt ist dabei die Kompatibilität mit OpenAI gym beziehungsweise gymnasium, welche die Einbindung in verschiedensten Umgebungen ermöglicht. In mehreren Arbeiten wurden diese Implementierungen bereits für das Trainieren einer geeigneten policy verwendet und Vergleiche der Algorithmen innerhalb des Frameworks gezogen. Es stellt sich jedoch die Frage, ob die Leistung dieser Framework Lösungen mit einer speziell auf die jeweilige Umgebung (in diesem Fall Duckietown) zugeschnittenen Implementierung vergleichbar ist.

Dieser Frage wurde im Rahmen dieser Arbeit in der virtuellen Duckietown-Umgebung nachgegangen. Ziel war es, mögliche Qualitätsunterschiede zwischen der Lösung des Frameworks Stable Baselines 3 und der eigens entwickelten Lösung zu ermitteln. Hierfür wurden mittels DDPG sowie TD3, CNN-policies für die Einhaltung der rechten Fahrspur auf zwei verschiedenen Karten in der Duckietown-Umgebung trainiert. Anschließend wurde eine Evaluation der Ergebnisse mittels vier verschiedener Karten durchgeführt, um Unterschiede hinsichtlich des Fahrverhaltens, der erhaltenen Belohnung sowie der Anpassungsfähigkeit zwischen Stable Baselines 3 und der eigens erstellten Lösung festzustellen.

Unterschiede dieser Lösungen sind hinsichtlich der verwendeten Zustandsvorverarbeitung (Bildgröße, Farbraum, Normalisierung der Farbwerte) sowie in der Implementierung des Aktionsrauschens festzustellen. So verwendet Stable Baselines 3 zum einen eine Vektorisierung der Umgebung und zum anderen ein zustandsabhängiges Aktionsrauschen. Dahingegen wurde in der eigens erstellten Lösung eine Komprimierung der Zustandsinformationen hinsichtlich des Farbraums sowie eine Normalisierung der Werte vorgenommen. Darüber hinaus erfolgte hier die Implementierung eines zustandsunabhängigen Aktionsrauschens. Gemeinsamkeiten der eigens erstellten Lösung und dem Framework Stable Baselines 3 sind in der Verwendung des deep learning Frameworks Pytorch und durch den Einsatz von gymnasium als Schnittstelle zu der Simulationsumgebung festzustellen.

Die Betrachtungen während und nach dem Training zeigten zunächst einige Parallelen beider Implementierungen. So war TD3 in beiden Fällen in der Lage, eine bessere Leistung als DDPG zu zeigen. Dies galt neben der erreichten Belohnungsmenge in weiten Teilen auch für die Fahrzeuggeschwindigkeit und die Ausrichtung des Fahrzeuges innerhalb der Fahrspur. Ebenso konnten stabile konvergierende Verhalten hinsichtlich der Belohnung festgestellt werden. Nach Abschluss des Trainings, waren die policies beider Implementierungen in der Lage mit dem Duckiebot, die trainierte Strecke in Duckietown abzufahren. Jedoch konnten leichte ruckartige Lenkbewegungen festgestellt werden. Darüber hinaus zeigten beide Implementierungen Schwankungen, welche sich unterschiedlich äußerten. So schwankte der Belohnungungsverlauf bei Betrachtungen der erstellten Implementierung teils stark. Dies führte in einigen Fällen zu einem Abfallen der Belohnung kurz vor dem Erreichen des Trainingsendes. Dahingegen sind bei Stable Baselines 3 Schwankungen über die Grenzen eines Trainingslaufes hinaus zu sehen. Diese führten in einigen Fällen zu

einem Konvergieren bei einer geringen Belohnung, was einen erneuten Trainingslauf erforderte. Der erzeugte Graph der Belohnung zeigte jedoch nur sehr geringen Schwankungen. Dies kann als Vorteil gewertet werden, da so das erreichte Leistungsniveau besser gehalten werden kann.

Unterschiede zwischen der eigens erstellten Lösung und Stable Baselines 3 wurden insbesondere bei Betrachtungen in neuen Umgebungen, wie nicht bereits aus dem Training bekannten Karten, festgestellt. Hier war die eigens erstellte Lösung dieser Arbeit in der Lage, in einem leicht veränderten Umfeld meist gute Metriken zu erreichen. Dieses Verhalten konnte bei einer veränderten Gestaltung der Bedingungen neben der Strecke als auch bei Veränderungen hinsichtlich der Streckenführung festgestellt werden. Herausfordernd blieben hier besonders kurvenreiche Streckenführungen. Die Anpassung auf neue Streckenführungen war insbesondere bei TD3 zu erkennen. Die mittels Stable Baselines 3 trainierten policies zeigten in vielen Bereichen ein Verhalten der Überanpassung. So führten Veränderungen des Umfeldes zu unerwünschtem Verhalten sowie einem starken Abfallen der Belohnung. Dies betraf den Aspekt der Streckenführung, aber auch die Bedingungen im direkten Umfeld der Straße. So führte der Wechsel von einem Grünstreifen zu einer asphaltierten Fläche neben der Straße bereits zu einem deutlichen Leistungsunterschied. Problematisch ist dieses Verhalten insbesondere bei einer Übertragung der Lösung des Frameworks in die reale Duckietown oder den realen Straßenverkehr. Durch die Interaktion mit anderen Verkehrsteilnehmer:innen sowie die wechselnde Umgebung treten hier häufig neue Situationen auf, welche für eine problematisch solche Überanpassung sind. Deutliche Vorteile sind allerdings im Bereich der Laufzeitoptimierung von Stable Baselines 3 festzustellen. Diese ermöglichten eine um etwa 50% verkürzte Trainingsdauer im direkten Vergleich. So erweist sich in kurzen Trainingsläufen und bekannten Strecken die Anwendung von Stable Baselines 3 als sinnvoll, in unbekannten Kontexten hingegen ist die eigens entwickelte Lösung leistungsfähiger.

Anhand dieser Ergebnisse kann die Eingangs gestellte Frage: „Welcher Qualitätsunterschied lässt sich zwischen der allgemein anwendbaren Lösung des Frameworks Stable Baselines 3 und der eigens für die Duckietown-Umgebung entwickelten Lösung feststellen?“ in Abhängigkeit des verwendeten Anwendungsfalles betrachtet werden. So ist bei einer Anwendung der policies auf unterschiedlichen Karten und Umgebungen die im Rahmen dieser Arbeit erstellte Lösung vorzuziehen. Hierfür sprechen insbesondere das dargestellte Verhalten auf neuen Strecken sowie Streckenumgebungen. Für eine erste Analyse des Fahrverhaltens eines Duckiebots sowie für Vergleiche mit alternativen Steuerungsansätzen (policies) empfiehlt es sich hingegen, auf Stable Baselines 3 zurückzugreifen. Hierdurch könnten durch die kurzen Trainingslaufzeiten und guten Leistungen in bekannten Umgebungen schneller gute Ergebnisse erzielt werden.

Allerdings bleibt hierbei fraglich, inwieweit sich das in der Simulation beobachtete Verhalten bei einer Ausführung auf einem realen statt virtuellen Duckiebot widerspiegeln wird. Hier sind durchaus einige Unterschiede zu erwarten, da in der Simulation Aspekte wie Schattenwurf, Überbelichtungen oder beschädigte Fahrbahnmarkierungen nicht abgebildet werden. Die Optimierungen, hinsichtlich der Trainingsparameter sowie Zustandsvorverarbeitungen, im Rahmen dieser Arbeit repräsentieren außerdem nicht zwangsläufig das erreichbare Leistungsmaximum beider Implementierungen. Denkbar wäre beispielsweise, dass unter Verwendung anderer Parameter bessere Ergebnisse bei beiden Implementierungen erzielt werden könnten. Dies betrifft zum einen die Trainingsparameter, aber auch die Vorverarbeitung der Zustandsinformationen, welche weiter optimiert werden könnten.

Neben der Übertragung der Ergebnisse in eine reale Umgebung wäre die Implementierung eines zustandsabhängigen Aktionsrauschens in der eigens entwickelten Lösung von Interesse. Dieses könnte die leichten ruckartigen Lenkbewegungen etwas reduzieren und das Fahrverhalten stabilisieren. Darüber hinaus könnte die thematisierte Überanpassung seitens Stable Baselines 3 durch die Verwendung semantischer Segmentierung verringert werden. Hierfür wäre beispielsweise der Einsatz aktueller KI-Modelle, wie YOLOv11-seg, möglich. Dies könnte insbesondere die Leistung in neuen Umgebungen aufgrund der geringeren Unterschiede, steigern und so die

Leistung von Stable Baselines 3 verbessern. Ebenso könnte für die Wahl der Trainingsparameter Optimierungsverfahren eingesetzt werden, um eine strukturierte Anpassung der verwendeten Werte an das Optimum zu ermöglichen. Die vorliegenden Ergebnisse verdeutlichen die zunehmende Bedeutung leistungsfähiger und zugleich anpassungsfähiger reinforcement learning-Modelle für den autonomen Fahrbetrieb. Insbesondere im Kontext aktueller Forschung zum Sim2Real-Transfer zeigt sich, dass robuste Generalisierungsfähigkeiten ein Schlüssel sind, um reale Verkehrssituationen effektiv abzubilden und eine sichere Fahrspurhaltung zu garantieren. In der langfristigen Perspektive lassen sich hieraus Möglichkeiten ableiten, die Zahl verkehrsbedingter Unfälle weiter zu reduzieren. Darüber hinaus kann die Verbindung effizienter Trainingsmethoden mit einer möglichst realitätsnahen Simulationsumgebung einen wesentlichen Beitrag zu nachhaltigen und ressourcenschonenden Ansätzen in der KI-Forschung leisten. Langfristig kann dies die Etablierung autonomer Fahrzeuge fördern und deren Potenziale, etwa im Hinblick auf Verkehrssicherheit, Effizienz und Umweltverträglichkeit, gezielter nutzbar machen.

Literatur

[1] KBA, „Anzahl zugelassener Pkw in Deutschland von 1960 bis 2024 (in 1.000),“ März 2024.

[2] Statistisches Bundesamt, „Anzahl der Straßenverkehrsunfälle in Deutschland von 1950 bis 2023,“ Juli 2024.

[3] M. V. Rajasekhar und A. K. Jaswal, „Autonomous vehicles: The future of automobiles,“ in *2015 IEEE International Transportation Electrification Conference (ITEC)*, IEEE, 82015, S. 1–6, ISBN: 978-1-5090-1911-3. DOI: 10.1109/ITEC-India.2015.7386874.

[4] S. Pischinger und U. Seiffert, Hrsg., *Vieweg Handbuch Kraftfahrzeugtechnik* (ATZ/MTZ-Fachbuch), 9., erweiterte und ergänzte Auflage. Wiesbaden und Heidelberg: Springer Vieweg, 2021, ISBN: 9783658255565.

[5] G. Bathla u. a., „Autonomous Vehicles and Intelligent Automation: Applications, Challenges, and Opportunities,“ *Mobile Information Systems*, Jg. 2022, S. 1–36, 2022, ISSN: 1574-017X. DOI: 10.1155/2022/7632892.

[6] W. Shi, G. Huang, S. Song, Z. Wang, T. Lin und C. Wu, *Self-Supervised Discovering of Interpretable Features for Reinforcement Learning*, 16. März 2020. Adresse: http://arxiv.org/pdf/2003.07069v4.

[7] PyPi Stats, *torchrl*. besucht am 13. Apr. 2025. Adresse: https://pypistats.org/packages/torchrl.

[8] A. Raffin, J. Kober und F. Stulp, „Smooth Exploration for Robotic Reinforcement Learning,“ *Proceedings of the 5th Conference on Robot Learning*, Adresse: http://arxiv.org/pdf/2005.05719v2.

[9] PyPi Stats, *stable-baselines3*. besucht am 13. Apr. 2025. Adresse: https://pypistats.org/packages/stable-baselines3.

[10] Florian Pillau, *Autonomes Fahren: Mercedes wird Level-3-autonomes Fahren bis 95 km/h gestattet*, heise, Hrsg., 2024. besucht am 10. Apr. 2025. Adresse: https://www.heise.de/news/Autonomes-Fahren-Mercedes-wird-Level-3-autonomes-Fahren-bis-95-km-h-gestattet-10207389.html.

[11] waymo, *waymo Website*. besucht am 10. Apr. 2025. Adresse: https://waymo.com.

[12] Duckietown, *Duckietown Mission*. besucht am 22. Dez. 2024. Adresse: https://duckietown.com/mission/.

[13] *Duckiebot*. besucht am 10. Apr. 2025. Adresse: https://at.rs-online.com/web/p/schulungs-erfindungskits/2304507.

[14] *Duckietown physisch*. besucht am 10. Apr. 2025. Adresse: https://www.hdmotori.it/2018/11/05/guida-autonoma-olimpiadi-auto-intelligenti/.

[15] Duckietown, *Duckietown Technlogy Platform*. besucht am 22. Dez. 2024. Adresse: https://duckietown.com/platform/.

[16] M. Chevalier-Boisvert, F. Golemo, Y. Cao, B. Mehta und L. Paull, *Duckietown Environments for OpenAI Gym*, 2018.

[17] Valentin Braitenberg, „Vehikel: Experimente mit künstlichen Wesen,“ 2004.

[18] N. Ahmed und W. J. Teahan, „Using Compression to Discover Interesting Behaviours in a Hybrid Braitenberg Vehicle," *IEEE Access*, Jg. 9, S. 11 316–11 327, 2021. DOI: 10.1109/ACCESS.2021.3050882.

[19] I. Rano, „A model and formal analysis of Braitenberg vehicles 2 and 3," in *2012 IEEE International Conference on Robotics and Automation*, IEEE, 52012, S. 910–915, ISBN: 978-1-4673-1405-3. DOI: 10.1109/ICRA.2012.6224583.

[20] S. NMascetti, D. Ahmetovic, A. Gerino, C. Bernareggi, M. Busso und A. Rizzi, „Robust traffic lights detection on mobile devices for pedestrians with visual impairment," *Computer Vision and Image Understanding*, Jg. 148, S. 123–135, 2016, ISSN: 10773142. DOI: 10.1016/j.cviu.2015.11.017.

[21] Aisha Ajmal, Christopher Hollitt, Marcus Frean und Harith Al-Sahaf, *2018 International Conference on Image and Vision Computing New Zealand (IVCNZ)*. Piscataway, NJ: IEEE, 2018, ISBN: 9781728101255. Adresse: http://ieeexplore.ieee.org/servlet/opac?punumber=8625178.

[22] Kayla Cameron und Md Shafiul Islam, *Proceedings of the 2018 IEEE National Aerospace and Electronics Conference (NAECON)*. Piscataway, NJ: IEEE, 2018, ISBN: 9781538665572. Adresse: http://ieeexplore.ieee.org/servlet/opac?punumber=8540799.

[23] Aishwarya Singh, *9 Powerful Tricks for Working with Image Data using skimage in Python*. besucht am 14. Apr. 2025. Adresse: https://medium.com/analytics-vidhya/9-powerful-tricks-for-working-with-image-data-using-skimage-in-python-a84c3656663d.

[24] C. Junhua und L. Jing, „Research on Color Image Classification Based on HSV Color Space," in *2012 Second International Conference on Instrumentation, Measurement, Computer, Communication and Control*, IEEE, 122012, S. 944–947, ISBN: 978-1-4673-5034-1. DOI: 10.1109/IMCCC.2012.226.

[25] R.Surya Kumar, P.Kingston Stanley, P.Kingston Stanley, *Raspberry Pi Based Vehicle Collision Avoidance System: ´*. Piscataway, NJ: IEEE, 2017, ISBN: 9781509045273. Adresse: http://ieeexplore.ieee.org/servlet/opac?punumber=8107622.

[26] user7859, *Submission 20856*, 2023. Adresse: https://challenges.duckietown.org/v4/humans/submissions/20856.

[27] S. H. Ashwin und R. Naveen Raj, „Deep reinforcement learning for autonomous vehicles: lane keep and overtaking scenarios with collision avoidance," *International Journal of Information Technology*, Jg. 15, Nr. 7, S. 3541–3553, 2023, ISSN: 2511-2104. DOI: 10.1007/s41870-023-01412-6.

[28] E. H. Sumiea u. a., „Deep deterministic policy gradient algorithm: A systematic review," *Heliyon*, Jg. 10, Nr. 9, e30697, 2024, ISSN: 2405-8440. DOI: 10.1016/j.heliyon.2024.e30697.

[29] S. Aradi, „Survey of Deep Reinforcement Learning for Motion Planning of Autonomous Vehicles," *IEEE Transactions on Intelligent Transportation Systems*, Jg. 23, Nr. 2, S. 740–759, 2022, ISSN: 1524-9050. DOI: 10.1109/TITS.2020.3024655.

[30] R. S. Sutton und A. Barto, *Reinforcement learning: An introduction* (Adaptive computation and machine learning), Second edition. Cambridge, Massachusetts und London, England: The MIT Press, 2020, ISBN: 9780262039246.

[31] K. Arulkumaran, M. P. Deisenroth, M. Brundage und A. A. Bharath, „Deep Reinforcement Learning: A Brief Survey," *IEEE Signal Processing Magazine*, Jg. 34, Nr. 6, S. 26–38, 2017, ISSN: 1053-5888. DOI: 10.1109/MSP.2017.2743240.

[32] P. Miralles, „From simulation to reality: From simulation to reality Study and demonstration of domain adaptation techniques applied to Reinforcement and Supervised learning algorithms.," Master Thesis, ISAE Supaero, Frankreich, 2020-11-19.

[33] Scott Fujimoto, Herke van Hoof und David Meger, „Addressing Function Approximation Error in Actor-Critic Methods,".

[34] X. Wang u. a., „Deep Reinforcement Learning: A Survey," *IEEE transactions on neural networks and learning systems*, Jg. 35, Nr. 4, S. 5064–5078, 2024. DOI: 10.1109/TNNLS.2022.3207346.

[35] Kalapos András, Gór Csaba, Moni Róbert und Harmati István, „Vision-based reinforcement learning for lane-tracking control,".

[36] Greg Brockman u. a., *OpenAI Gym*, 2016.

[37] M. Towers u. a., „Gymnasium: A Standard Interface for Reinforcement Learning Environments," *arXiv preprint arXiv:2407.17032*, 2024.

[38] D. Li und O. Okhrin, „A platform-agnostic deep reinforcement learning framework for effective Sim2Real transfer towards autonomous driving," *Communications engineering*, Jg. 3, Nr. 1, S. 147, 2024. DOI: 10.1038/s44172-024-00292-3.

[39] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus und Noah Dormann, „Stable-Baselines3: Reliable Reinforcement Learning Implementations," *Journal of Machine Learning Research*, Jg. 22, Nr. 268, S. 1–8, 2021. Adresse: http://jmlr.org/papers/v22/20-1364.html.

Abbildungsverzeichnis

Tabellenverzeichnis

Listingverzeichnis

Abkürzungsverzeichnis

PKW Personenkraftwagen

KI künstliche Intelligenz

TD3 Twin Delayed Deep Determenistic Policy Gradient

DDPG Deep Determenistic Policy Gradient

VDA Verband der Automobilindustrie

MOOC massive open online course

IMU inertiale Messeinheit

SAC Soft Actor Critic

Anhang

1 Ergebnisse der Evaluation in verschiedenen Umgebungen

		Eigens implementiert -loop empty simple-		**Eigens implementiert -loop empty-**		**Stable Baselines 3**	
Umgebung	**Metrik**	DDPG	TD3	DDPG	TD3	DDPG	TD3
loop empty	Mittlere Belohnung	115.32	276.019	732.74	448.73	264.09	254.40474
	Mittlere Fahrzeuggeschwindigkeit	0.12	0.53	0.34	0.52	0.29	0.35
	Winkel Belohnung	0.04	0.02	0.34	0.33	0.18	0.25
	Mittlere Dauer je Episode	15.42	14.09	15.20	7.30	7.02	6.54
4 way	Mittlere Belohnung	920.58	470.24	164.48	438.66	29.58	32.39
	Mittlere Fahrzeuggeschwindigkeit	0.77	0.71	0.46	0.78	0.57	0.63
	Winkel Belohnung	0.43	0.33	0.24	0.29	0.14	0.13
	Mittlere Dauer je Episode	11.30	5.49	2.53	5.60	0.70	0.56
zig zag	Mittlere Belohnung	103.1	30.87	49.96	19.95	26.13	21.02
	Mittlere Fahrzeuggeschwindigkeit	0.29	0.41	0.31	0.52	0.38	0.46
	Winkel Belohnung	0.06	0.13	0.10	0.03	0.08	0.06
	Mittlere Dauer je Episode	9.66	3.28	1.73	0.89	6.35	0.95

Tabelle 2: Ergebnisse in verschiedenen Umgebungen

Erklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbstständig verfasst und dabei keine anderen als die angegebenen Hilfsmittel benutzt habe. Sämtliche Stellen der Arbeit, die im Wortlaut oder dem Sinn nach Werken anderer Autoren entnommen sind, habe ich als solche kenntlich gemacht. Die Arbeit wurde bisher weder gesamt noch in Teilen einer anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

14. April 2025

Tim Halle