Experiment zur Verbesserung der Robustheit von Reinforcement Learning Modellen in Drohnensimulationen anhand trainiertem Gegenspieler

Bachelorarbeit

vorgelegt am 22. April 2023

Fakultät Wirtschaft

Studiengang Wirtschaftsinformatik

Kurs WWI2020F

von

Leon Henne

Betreuerin in der Ausbildungsstätte:	DHBW Stuttgart:
IBM Deutschland GmbH Sophie Lang Senior Data Scientist	Prof. Dr. Kai Holzweißig Studiendekan Wirtschaftsinformatik
Unterschrift der Betreuerin	

Inhaltsverzeichnis

Abkürzungsverzeichnis

DHBW Duale Hochschule Baden-Württemberg

IL Imitation Learning

RL Reinforcement Learning

KPI Key Performance Indicator

MARL Multi-Agent Reinforcement Learning

DART Dynamic Animation and Robotics Toolkit

ODE Open Dynamics Engine

ROS Robot Operating System

SITL Software-in-the-Loop

RAEARL Robust Adaptive Ensemble Adversarial Reinforcement Learning Framework

DQL Deep Q-Learning

TRPO Trust Region Policy Optimization

PPO Proximal Policy Optimization

A3C Asynchronous Advantage Actor-Critic

A2C Advantage Actor-Critic

DR Domain Randomization

CSV kommaseperierte Wertedatei

Abbildungsverzeichnis

Tabellenverzeichnis

[1] Ivaldi/Padois/Nori 2014, S. 4
[2] Pullum 2022, S.17
[3] Pullum 2022, S.19

1 Einleitung

1.1 Problemstellung

Reinforcement Learning (RL) findet heutzutage bereits Anwendung in vielerlei Forschungsprojekten wie Deepmind AlphaStar oder OpenAI Five, aber auch in Produkten und Dienstleistungen wie AWSDeepRacer oder Meta's Horizon open-source RL-Plattform.[4] RL ist im Bereich des maschinellen Lernens eine Herangehensweise zur Lösung von Entscheidungsproblemen.[5] Ein Software-Agent leitet dabei durchzuführende Aktionen aus seiner Umgebung ab, mit dem Ziel die kumulierte erhaltene Belohnung zu maximieren, währenddessen sich seine Umgebung durch alle Aktionen verändert.[6] Die Umgebungen beinhalten in ihrer einfachsten Form eine simulierte Welt, welche zu jedem Zeitschritt eine Aktion entgegennimmt, und den eigenen nächsten Zustand sowie einen Belohnungswert zurückgibt.[7] Da das Sammeln von Daten in der echten Welt für den Einsatz von RL Algorithmen eine Limitierung darstellen kann, werden häufig fürs Training Simulationsumgebungen eingesetzt[8] Eine Limitierung können bspw. Sicherheitsaspekte sein, welche beim Training von Roboterarmen, oder sich autonom bewegenden Systemen auftreten, da die einzelnen physischen Bewegungen nicht vorhersehbar abschätzbar sind.[9] Simulationen nehmen damit zum einen als Testumgebung eine wichtige Rolle in der Entwicklung von Kontrollalgorithmen ein.[10] Zum anderen bedarf die erfolgreiche Anwendung von RL neben effizienten Algorithmen auch geeignete Simulationsumgebungen.[11] Besonders schwierig, und daher sehr wichtig zu erforschen, ist die Anpassung der Trainingsumgebung an die reale Welt, sodass bspw. Agenten für Roboter und autonome Fahrzeuge nach dem Training mit robusten Strategien in der Realität eingesetzt werden können.[12] In der Forschungsliteratur wird diese beschriebene Problematik als „Sim to real"-Transfer beschrieben.[13]

Ein Forschungsgebiet, bei dem die Lösung des Sim to Real Transfers betrachtet wird, ist die autonome Steuerung von unbemannten Luftfahrzeugen bzw. Drohnen.[14] Das Transferproblem entsteht bspw. dabei, dass zur automatisierten Kollisionsvermeidung die Kollisionsbeispiele einer Simulationsumgebung entnommen werden, um physischen Schaden oder Drohnenverlust zu vermeiden.[15] Drohnen tragen dabei bereits in der heutigen Zeit zur Lösung vieler komplexer Aufgaben bei, wie der Katastrophenüberwachung oder der Waldbrandbekämpfung.[16] Zusätzlich

[4] Vgl. Li 2019, S. 4
[5] Vgl. Schuderer/Bromuri/van Eekelen 2021, S. 3
[6] Vgl. Schuderer/Bromuri/van Eekelen 2021, S. 3
[7] Vgl. Reda/Tao, T./van de Panne 2020, S. 1
[8] Vgl. Zhao/Queralta/Westerlund 2020, S. 737
[9] Vgl. Zhao/Queralta/Westerlund 2020, S. 738
[10] Vgl. Cutler/Walsh/How 2014, S. 2
[11] Vgl. Reda/Tao, T./van de Panne 2020, S. 8
[12] Vgl. Slaoui u. a. 2019, S. 1
[13] Vgl. Zhao/Queralta/Westerlund 2020, S. 738
[14] Vgl. Deshpande/Minai/Kumar, M. 2021, S. 1
[15] Vgl. Sadeghi/Levine 2016, S. 4
[16] Vgl. Hentati u. a. 2018, S. 1495

steigen immer weiter die komplexen Einsatzanforderungen unter dem Anstieg an Anwendungsgebieten für unbemannte Luftfahrzeuge.[17]

Die Simulation einer möglichst realistischen Umgebung in diesem Kontext wird in der Forschung häufig mit dem Ansatz von Domain Randomization (DR) begleitet.[18] Unter dem Themenfeld der DR wird die Idee erforscht, anstelle der akkuraten Modellierung realistischer Dynamiken, diese so stark zu randomisieren, dass reale Dynamikeffekte abgedeckt sind.[19] Neben den dynamischen Bedingungen unterliegt die Realität jedoch häufig auch dem Einfluss mehrerer Parteien. Diese tragen teilweise kooperierend aber auch teilweise konkurrierend zum eigenen Erfolg bei, wie z.B. im Rahmen eines dem Wettbewerb unterliegenden Marktes.[20] Stellt man sich ein Szenario im Kontext kooperativer oder konkurrierender Drohnen vor, ist es naheliegend, dass auch jene Einflüsse möglichst präzise in die Simulationsumgebung integriert sein müssen, um ein robustes Modell erlernen zu können. Während bereits in Produkten wie PowerTAC von Collins/ Ketter 2022 die Simulation von Märkten entwickelt wurde, scheint der Einfluss des Gegenspielers in kompetitiven Drohnensimulationen auf die Robustheit von RL Algorithmen, und demnach auf die Lösung des „Sim to real"-Transfers, unerforscht.

1.2 Zielsetzung

Im Rahmen dieser Arbeit soll eine Simulationsumgebung entwickelt werden, in welcher sich RL basierter Gegenspieler und deterministische Gegenspieler integrieren lassen. Anschließend ist zu Untersuchen, wie sich die Wahl des Trainingsgegenspielers auf das Leistungsverhalten der RL Modelle Verhaltensmodelle, im Kontext von RL oftmals als Policies referenziert, unter verändertem Testszenario auswirkt.

Dazu soll eine kompetitive Simulationsumgebung entwickelt werden, in welcher sich zwei konkurrierender Spieler in Form von Flugobjekten spielerisch gegenseitig bekämpfen. In der Simulation werden folgende Policies in drei verschiedenen Szenarien trainiert.

- Training mit regelbasiertem Gegenspieler unter gleichbleibenden Dynamikparametern
- Training mit RL basiertem Gegenspieler unter gleichbleibenden Dynamikparametern
- Training mit regelbasiertem Gegenspieler unter sich verändernden Dynamikparametern

Anschließend werden alle trainierten Policies in einem Testszenario untersucht. Das Testszenario verfügt dabei über festgelegte sich vom Training unterscheidende Eigenschaften, wie Flugbahnen, oder Start- und Zielpositionen. Außerdem wird ein deterministischen Gegenspieler unter im Gegensatz zum Training, veränderten Handlungspräferenzen eingesetzt. Bei der Untersuchung werden jeweils die folgenden Variablen als Key Performance Indicator (KPI) betrachtet.

[17] Vgl. Deshpande/Kumar, R. u. a. 2020, S. 1
[18] Vgl. Sadeghi/Levine 2016, S. 1
[19] Vgl. Zhao/Queralta/Westerlund 2020, S. 4f.
[20] Vgl. Collins/Ketter 2022, S. 2

- kumulierte erzielte Belohnung
- Varianz der Belohnungen
- Anzahl an Misserfolgen

Durch die Auswertung des Testszenarios kann der Effekt des RL basierten Gegenspielers auf die Robustheit evaluiert werden, indem die Leistungsdiskrepanz zwischen Trainings- und Testszenario mit der des regelbasierten Gegenspieler und der der Domain Randomization verglichen wird.

1.3 Forschungsfrage

Aus der beschriebenen Problemstellung und der für den Rahmen dieser Arbeit festgelegten Zielsetzung ergibt sich folgende Forschungsfrage:

Inwiefern kann durch den Einsatz eines mittels RL trainierten Gegenspielers die Robustheit der gelernten Policy verbessert werden?

Zur Beantwortung der Forschungsfrage werden folgende Hypothesen aufgestellt und im Rahmen der Arbeit untersucht:

Hypothese 1: *Die im Testszenario erzielte kumulierte Belohnung ist unter Verwendung der Policy aus dem Training mit RL basiertem Gegenspieler signifikant und zuverlässig höher als die Policy aus dem Training mit regelbasiertem Gegenspieler.*

Hypothese 2: *Die Varianz der im Testszenario erzielten kumulierten Belohnung ist unter Verwendung der Policy aus dem Training mit RL basiertem Gegenspieler signifikant und zuverlässig geringer als die Policy aus dem Training mit regelbasiertem Gegenspieler.*

Hypothese 3: *Die im Testszenario erreichte Anzahl von Misserfolgen ist unter Verwendung der Policy aus dem Training mit RL basiertem Gegenspieler signifikant und zuverlässig geringer als die Policy aus dem Training mit regelbasiertem Gegenspieler.*

1.4 Forschungsmethodik

Als Forschungsmethodik soll im Rahmen dieser Arbeit ein quantitatives Laborexperiment nach Recker 2021 durchgeführt werden. Hierbei wird häufig nach dem hypothetisch-deduktiven Modell vorgegangen, in welchem Hypothesen formuliert, empirische Studien entwickelt, Daten gesammelt, Hypothesen anhand dieser evaluiert und gewonnene Erkenntnisse berichtet werden.[21] Damit stellt das Laborexperiment eine Möglichkeit der Untersuchung der Ursache- und Wirkungsbeziehung dar.[22] Dabei wird die kontrollierte Umgebung der Simulation erschaffen, deren

[21] Vgl. Recker 2021, S. S.89f.
[22] Vgl. Recker 2021, S. 106

Aufbau die unabhängige Variable des Laborexperiments darstellt. Die Metriken, anhand der die Leistung und die Robustheit der trainierten Policies gemessen werden, bilden im Experiment die abhängigen Variablen, zu dessen Grundlage die Forschungsfrage beantwortet wird.

1.5 Aufbau der Arbeit

Insgesamt gliedert sich die Arbeit nach einem Schema von Holzweißig 2022 Die Arbeit beginnt mit einem einleitenden Kapitel, in welchem Motivation, Problemstellung, Zielsetzung und Forschungsmethodik erläutert sind. Anschließend wird im zweiten Kapitel der aktuelle Stand der Forschung zu den relevanten Konzepten beschrieben. Im dritten Kapitel wird die Forschungsmethodik dargestellt, indem die Simulationsumgebung als Messinstrument entwickelt wird, verschiedene Messszenarien erläutert und entsprechende Daten gesammelt werden. Daraufhin erfolgt im vierten Kapitel die Auswertung der Messdaten sowie die Beantwortung der Forschungsfrage durch Annahme oder Ablehnung der eigens aufgestellten Hypothesen. Im Zuge dessen kann ebenso die Forschungsfrage anhand der Annahme oder Ablehnung der Hypothesen beantwortet werden. Abschließend wird im letzten Kapitel ein Fazit zu den erzielten Forschungsergebnissen dargelegt und ein Ausblick auf nachfolgende Forschungsansätze gegeben.

2 Diskussion des aktuellen Stands der Forschung und Praxis

2.1 Aufbau der Literaturrecherche

In Anlehnung an die Literaturrecherche nach Webster/Watson 2002 wurden alle voraussichtlich benötigten Konzepte für die Durchführung der beschriebenen Forschungsmethodik in Tabelle 1 festgehalten. Alle angeführten Konzepte wurden mittels verschiedener Suchbegriffe in Suchmaschinen, Datenbanken und Bibliotheken wie *Google Scholar* 2/28/2023, *IEEE Xplore* 2/28/2023 oder die digitale Bibliothek der Association for Computing Machinery (ACM) ACM Digital Library 2/28/2023 recherchiert. In der daraus gefunden Literatur wurden zitierte Werke ebenfalls nach den beschriebenen Konzepten durchsucht und insgesamt jede Literaturquelle in Tabelle 1 den in ihnen enthaltenen Konzepten zugeordnet.

Artikel	Konzepte							
	RL	MARL	IL	ES	DS	KS	DR	RRLP
Sutton/Barto 2018	X							
Li 2019	X							
Zhao/Queralta/Westerlund 2020	X			X			X	
Wang/Hong 2020	X							
Zhang/Wu/Pineau 2018	X			X				X
Cutler/Walsh/How 2014	X			X				
Canese u. a. 2021	X	X						
Reda/Tao, T./van de Panne 2020	X			X			X	
Ningombam 2022	X							
Arulkumaran u. a. 2017	X							
Huang u. a. 2017	X							
Mnih/Kavukcuoglu u. a. 2013	X							
Wong u. a. 2022	X	X						
Hussein u. a. 2017	X		X					
Attia/Dayan 2018			X					
Gao u. a. 2014			X					
Fang u. a. 2019			X					
Balakrishna u. a. 2020			X					
Schuderer/Bromuri/van Eekelen 2021	X	X		X				
Körber u. a. 2021				X				
Bharadhwaj u. a. 2019				X			X	
Foronda 2021				X				
Maria 1997				X				

Artikel	Konzepte							
Brockman u. a. 2016	X			X				
Yan Duan u. a. 2016	X			X				X
Ivaldi/Padois/Nori 2014				X				
Ayala u. a. 2020				X				
Todorov/Erez/Tassa 2012				X				
Koch u. a. 2018	X				X			
Deshpande/Kumar, R. u. a. 2020	X				X			
Deshpande/Minai/Kumar, M. 2021					X		X	X
Hentati u. a. 2018					X			
Molchanov u. a. 2019					X		X	X
Furrer u. a. 2016					X			
Silano/Iannelli 2019					X			
Shah u. a. 2017					X			
Panerati u. a. 2021				X	X			
Moos u. a. 2022								X
Pullum 2022								X
Liu u. a. 2023								X
Yan Duan u. a. 2016								X
Schott/Hajri/Lamprier 2022	X					X		
Pinto u. a. 2017	X					X		X
Pan u. a. 2021						X		
Zhai u. a. 2022						X		X
Tobin u. a. 2017							X	
Chen u. a. 2021							X	
Hsu u. a. 2023							X	
Alghonaim/Johns 2021							X	
Sadeghi/Levine 2016							X	

Tab. 1: Konzeptmatrix für Artikel zu Simulationsumgebungen und zur Robustheit RL Algorithmen nach Webster/Watson 2002. Legende: RL (Reinforcement Learning), MARL (Multi-Agent Reinforcement Learning), IL (Imitation Learning), ES (Entwicklung von Simulationsumgebungen), DS (Drohnensimulation), KS (kompetitive Simulationsumgebungen), DR (Domain Randomization), RRLP (Robustheit von RL Policies)

2.2 Verstärkendes Lernen

Verstärkendes Lernen, oder auch als Reinforcement Learning (RL) in der Fachsprache bezeichnet, definiert einen konzeptionellen Ansatz zielorientiertes Lernen von Entscheidungen zu verstehen

und zu automatisieren.[23] Dabei besteht der Fokus darauf, dass ein Agent aus der direkten Interaktion mit seiner Umgebung lernt, ohne dass explizite Überwachung notwendig ist.[24] Der Agent lernt über die Zeit eine optimale Strategie zur Lösung des Entscheidungsproblems durch Ausprobieren und Scheitern die gewünschte Veränderung in seiner Umwelt herzustellen.[25] Notwendig dabei ist es, dass der Agent den Zustand seiner Umgebung wahrnehmen und durch entsprechende Aktionen beeinflussen kann, um das Erreichen des Zielzustandes zu ermöglichen.[26] Dazu muss der Agent diejenigen Aktionen finden, die ihm die größtmögliche kumulierte Belohnung liefern, wobei Aktionen nicht nur die unmittelbaren sondern auch zukünftige Belohnungen beeinflussen.[27] Zusammengefasst lässt sich die beschriebene Interaktion des Agenten mit seiner Umgebung wie in Abbildung Eins darstellen.

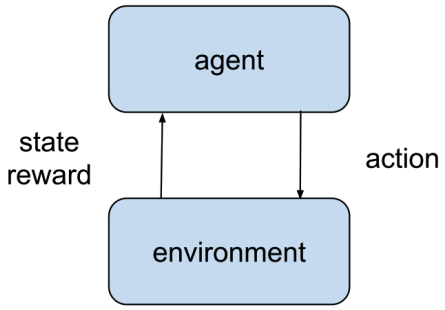


Abb. 1: vereinfachte Darstellung der Interaktion zwischen dem Agenten und seiner Umgebung[28]

Ein Standardaufbau einer Aufgabe für verstärkendes Lernen kann demnach als sequentielles Entscheidungsproblem verstanden werden, zu dessen Lösung ein Agent zu jedem diskreten Zeitschritt eine Aktion ausführt, welche den Zustand der Umgebung verändert.[29] Betrachtet man die technische Umsetzung einer solchen Interaktion zwischen dem Agenten und dessen Umgebung, wird häufig zur Modellierung ein Markov Entscheidungsprozess verwendet. Im Kontext von RL ist der Entscheidungsprozess definiert als ein Tupel aus folgenden Elementen:[30]

- Zustandsraum S
- Aktionsraum A
- initiale Zustandsverteilung $p_0(S)$
- Übergangswahrscheinlichkeit $T(S_{t+1}|S_t, A_t)$
- Belohnungswahrscheinlichkeit $R(r_{t+1}|S_t, A_t)$

[23] Vgl. Sutton/Barto 2018, S. 13
[24] Vgl. Sutton/Barto 2018, S. 13
[25] Vgl. Li 2019, S. 4
[26] Vgl. Sutton/Barto 2018, S. 2
[27] Vgl. Sutton/Barto 2018, S. 1
[28] Enthalten in: Li 2019, S. 5
[29] Vgl. Zhao/Queralta/Westerlund 2020, S. 2
[30] Vgl. Zhang/Wu/Pineau 2018, S. 2

Zum Finden der optimalen Strategie existieren modellbasierende und modellfreie Algorithmen des verstärkenden Lernens.[31] Bei modellbasierenden Algorithmen wird das Umgebungsverhalten, also die Übergangs- und Belohnungswahrscheinlichkeiten, als bekannt vorausgesetzt.[32] Als modellbasierender Algorithmus wird z.B. dynamische Programmierung eingesetzt, um mittels Strategieevaluation und Strategieiteration die optimale Strategie zu finden.[33] Unter modellfreien Algorithmen wird zwischen einen wertbasierten-, einen strategiebasierten und einen Akteur-Kritik-Ansatz unterschieden.[34] Der Agent im Kontext von modellfreien RL Methoden kennt dabei nur die Zustände S und die Aktionen A, jedoch nicht das Umgebungsverhalten T oder die Belohnungswahrscheinlichkeit R.[35] Fasst man die Klassifizierung der Algorithmen und Methoden von RL zusammen, lässt sie sich wie folgt darstellen:

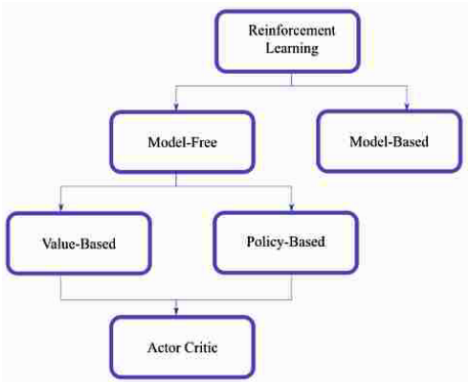


Abb. 2: Klassifizierung von Algorithmen im Bereich des RL[36]

2.2.1 Methoden des verstärkenden Lernens

Der Agent sucht im Kontext von **wertbasierenden Methoden** die optimale Strategie π^*, welche allen Zuständen S die jeweilige Aktion $A(S)$ zuordnet, sodass die kummulierte Belohnungswahrscheinlichkeit $R(r_{t+1}|S_t, A_t)$ über alle Zeitschritte t maximal ist.[37] Neben der unmittelbaren Belohnung aus einem Zustands-Aktions Paar müssen auch die zukünftigen Belohnungen der möglichen Folgezustände betrachtet werden, wofür das Konzept der Wertigkeit eingeführt wird.[38] Über eine Zustands- oder Aktionswertigkeitsfunktion, oftmals als Q-Funktion referenziert, wird eine Vorhersage über die zu erwartende abgezinste Belohnung berechnet.[39] Durch den Abzinsungsfaktor $\gamma \in [0, 1)$ wird der Einfluss zukünftiger Belohnungen nach ihrer zeitlichen Reihenfolge priorisiert.[40] Mit der Wertigkeitsfunktion kann evaluiert werden, welche Strategie langfristig am erfolgreichsten ist, da bspw. manche Aktionen trotz geringer sofortiger Belohnung

[31] Vgl. Wang/Hong 2020, S. 3
[32] Vgl. Wang/Hong 2020, S. 3
[33] Vgl. Li 2019, S. 5
[34] Vgl. Li 2019, S. 5
[35] Vgl. Cutler/Walsh/How 2014, S. 2
[36] Enthalten in: Canese u. a. 2021, S. 6
[37] Vgl. Reda/Tao, T./van de Panne 2020, S. 2
[38] Vgl. Wang/Hong 2020, S. 3
[39] Vgl. Li 2019, S. 5
[40] Vgl Li 2019, S. 5

einen hohen Wert aufweisen können, wenn aus dem zukünftigen Zustand eine hohe Belohnung zu erwarten ist.[41] Die Wertigkeitsfunktion und die daraus berechneten Wertigkeiten von Aktionen oder Zuständen werden über alle Zeitschritte neu geschätzt und stellen mit die wichtigsten Komponenten in Algorithmen des verstärkenden Lernens dar.[42] Methoden basierend auf diesem Wertigkeitswert lernen eine Schätzfunktion der Wertigkeit für alle Zustände ($V_\pi(s) \forall S$) und alle Zustandsaktions-Paare ($Q_\pi(s_t, a_t) \forall s, a \in (S, A)$) durch das Aktualisieren der folgenden Funktionen:[43]

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right] \quad (1)$$

$$V(s_t) = \max_a Q(s_t, a|\omega) \quad (2)$$

Aus der geschätzten Wertigkeit jedes Zustandsaktions Paares kann die optimale Strategie $\pi^*(s)$ durch $\arg\max_a Q(s, a)$ bestimmt werden.[44]

Methoden, welche die Strategie durch direkte Parametrisierung anstelle einer Bewertung aller Handlungsalternativen mittels Wertigkeitsfunktion optimieren, werden als **strategiebasierend** bezeichnet.[45] Diese Methodik kann beim Trainieren deterministischer Strategien zu unerwarteten Aktionen führen, weshalb häufig das Optimieren einer Wahrscheinlichkeitsverteilung für alle Aktionen bevorzugt wird.[46] Als Subklasse der RL Methoden wird der statistische Gradientenabstieg verwendet um die parametrisierte Strategie π_θ hinsichtlich der maximalen langfristigen kumulierten Belohnung zu optimieren.[47] Unter dem Parametervektor θ beschreibt die Strategie π_θ oder auch $\pi(a|s, \theta)$ die Wahrscheinlichkeit Aktion a im Zustand s auszuwählen.[48] Zur Optimierung der Strategie wird die Funktion der kumulierten Belohnungen J nach dem Parameter der Gewichte wie folgt in Formel drei abgeleitet und der optimierte Parametervektor anhand Formel vier aktualisiert.[49]

$$\nabla_\theta J(\theta) \approx \tfrac{1}{N} \sum_{i=1}^{N} \left[(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t))(\sum_{t=1}^{T} r(s_t, a_t)) \right] \quad (3)$$

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta) \quad (4)$$

Zusammengefasst können die Formeln vier und fünf dabei so interpretiert werden, dass die logarithmierte Wahrscheinlichkeit Aktion a_t im Zustand s_t auszuwählen erhöht wird, wenn a_t in einer höheren kumulierten Belohnung resultiert.[50]

[41] Vgl. Sutton/Barto 2018, S. 6
[42] Vgl. Sutton/Barto 2018, S. 6f.
[43] Vgl. Zhang/Wu/Pineau 2018, S. 2
[44] Vgl. Zhang/Wu/Pineau 2018, S. 2
[45] Vgl. Zhang/Wu/Pineau 2018, S. 2
[46] Vgl. Ningombam 2022, S. 3
[47] Vgl. Ningombam 2022, S. 3
[48] Vgl. Sutton/Barto 2018, S. 321
[49] Vgl. Wang/Hong 2020, S. 6
[50] Vgl. Wang/Hong 2020, S. 6

Unter **Akteur-Kritiker Methoden** werden hybride wertebasierende und strategiebasierende Methoden verstanden, welche zugleich die Strategie optimieren und eine Wertefunktion approximieren.[51] Die strategiebasierende Methodik mit der lernenden Strategie agiert dabei als Akteur, wohingegen die Wertefunktion, welche jeder Aktion und jedem Zustand einen Behlohnungswert zuweist, als Kritiker handelt.[52] Der Akteur wählt somit aus seiner Wahrscheinlichkeitsverteilung die auszuführende Aktionen aus, während der Kritiker diese anhand seiner Wertigkeit bewertet.[53] Betrachtet man den Trainingsprozess von Akteur-Kritiker basierten Methoden ist dieser wie folgt aufgebaut:[54]

1. Aktueller Zustand der Umgebung als Eingabe dem Akteur und Kritiker übergeben
2. Akteur liefert eine auszuführende Aktion basierend auf dem Umgebungszustand
3. Der Kritiker bekommt die Aktion als Eingabe und berechnet deren Wertigkeit mittels Q-Funktion
4. Durch die Wertigkeit seiner Aktion kann der Akteur seine Strategie anpassen
5. Mit der neuen Strategie führt der Akteur die nächste Aktion im folgenden Zustand aus
6. Die Q-Funktion des Kritikers wird mit den neuen Informationen aus der erhaltenen Belohnung angepasst

2.2.2 Algorithmen des verstärkenden Lernens

Im vorherigen Kapitel sind die Methoden des verstärkenden Lernens klassifiziert und deren Unterschiede beschrieben worden. Anschließend daran wird in diesem Abschnitt der Arbeit auf eine Auswahl konkreter Algorithmen und Implementierungen, die verschiedenen Methoden sowie deren Funktionsweise eingegangen. Die Auswahl an Algorithmen beinhaltet die fundamentalen Algorithmen des verstärkenden Lernens Deep Q-Learning (DQL), Trust Region Policy Optimization (TRPO) und Asynchronous Advantage Actor-Critic (A3C).[55]

Mit der Entwicklung von DQL konnte erstmals ein Algorithmus entwickelt werden, welcher eine Reihe von Atari 2600 Spielen auf der Fähigkeitsebene eines professionellen Videospieltesters spielen konnte.[56] Als wertbasierte Methode wird für jeden Zustand die Wertigkeit berechnet, was unter DQL mittels eines neuronalen Netzes erreicht wird, welches die Q-Funktion approximiert.[57] Durch Auswählen der Aktion mit der höchsten Wertigkeit in jedem Zustand, kann die deterministische Strategie abgeleitet werden.[58] DQL adressiert das Instabilitätsproblem von

[51] Vgl. Zhang/Wu/Pineau 2018, S. 2f.
[52] Vgl. Sutton/Barto 2018, S. 321
[53] Vgl. Ningombam 2022, S. 3
[54] Vgl. Ningombam 2022, S. 4
[55] Vgl. Arulkumaran u. a. 2017, S. 1
[56] Vgl. Arulkumaran u. a. 2017, S. 6
[57] Vgl. Huang u. a. 2017, S. 4
[58] Vgl. Huang u. a. 2017, S. 4

Funktionsapproximation unter dem Einsatz von Erfahrungswiederholung und Zielnetzwerken.[59] Erfahrungswiederholung beschreibt die Technik die Beobachtungen des Agenten, also in welchem Zustand welche Aktion zu welcher Belohnung und welchem Folgezustand führt, in jedem Zeitschritt zu speichern und zufällig anhand Übergangserfahrungen aus anderen Trainingsläufen die Gewichte der Wertigkeitsfunktion anzupassen.[60] Mittels Erfahrungswiederholung wird eine bessere Dateneffizienz zum einen durch die Wiederholung, und zum anderen durch das Auflösen von Korrelationen zwischen aufeinanderfolgenden Zuständen erzielt.[61] Das Zielnetzwerk beinhaltet die zunächst fixen Gewichte der Strategie, welche lediglich nach einer festen Anzahl an Schritten angepasst werden, um die Fluktuation der approximierten Q-Funktion auszugleichen.[62] Die Stärke von DQL liegt in der kompakten Repräsentation der Q-Funktion und der hochgradig dimensionierten Zustandsbeobachtungen durch neuronale Netze.[63]

Trust Region Policy Optimization (TRPO) ist ein strategiebasierender Gradientenalgorithmus zur effektiven Optimierung großer nicht linearer Strategien wie z.B. neuronale Netze.[64] Der Algorithmus weist dabei die zwei Varianten single-path, welche im modellfreien Kontext angewendet werden kann, und *vine* auf, welche sich nur in Simulationen eignet, da das System zu bestimmten Zuständen gespeichert wird.[65] Die Varianten unterscheiden sich im Schätzverfahren des Gradienten, wobei die single-path Methodik diesen anhand einer Aktions-Zustandskette der initialen Verteilung bestimmt, während unter *vine* eine Teilmenge verschiedener Aktion-Zustandspaare verwendet wird.[66] Die daraus entstehenden Anpassungen der Strategieparameter θ zwischen der alten und neuen Strategie werden mittels Kullback-Leibler-Divergenz bemessen und kontrolliert.[67] Eine Weiterentwicklung des TRPO Algorithmus ist Proximal Policy Optimization (PPO), durch dessen Verwendung die Implementierung erleichtert und die Datenprobenkomplexität verbessert wird.[68] PPO passt die Beschränkung der KL-Divergenz an, indem die Wahrscheinlichkeitsmasse der Optimierung außerhalb der gesetzten Hyperparametergrenzen bearbeitet, oder die Größe der KL-Divergenz bestraft wird.[69] Durch diese Anpassung der KL-Divergenz Beschränkung ermöglicht der PPO Algorithmus mehrfache Berechnungen des Gradienten anhand eines Datenobjektes.[70]

Zusätzlich zu den bisherigen Algorithmen kann durch jeweilige asynchrone Varianten, welche den Trainingsprozess des Agenten parallelisieren, die Stabilität des Trainings verbessert werden.[71] Einer dieser asynchronen Algorithmen im Bereich der Akteur-Kritiker Methodik stellt der asynchronous advantage actor-critic (A3C) Algorithmus dar, welcher anstelle von Erfahrungswiederholung

[59] Vgl. Arulkumaran u. a. 2017, S. 7
[60] Vgl. Mnih/Kavukcuoglu u. a. 2013, S. 4
[61] Vgl. Mnih/Kavukcuoglu u. a. 2013, S. 4f.
[62] Vgl. Arulkumaran u. a. 2017, S. 7
[63] Vgl. Arulkumaran u. a. 2017, S. 7
[64] Vgl. Schulman/Levine u. a. 2015, S. 1
[65] Vgl. Schulman/Levine u. a. 2015, S. 1
[66] Vgl. Schulman/Levine u. a. 2015, S. 4
[67] Vgl. Huang u. a. 2017, S. 4
[68] Vgl. Schulman/Wolski u. a. 2017, S. 1
[69] Vgl. Schulman/Wolski u. a. 2017, S. 3f.
[70] Vgl. Schulman/Wolski u. a. 2017, S. 4
[71] Vgl. Mnih/Badia u. a. 2016, S. 1

mehrere Agenten parallel in unterschiedlichen Umgebungsinstanzen trainiert.[72] A3C kombiniert die Berechnung der relativen Wertigkeit einer Aktion, anstelle der absoluten Wertigkeit durch die Q-Funktion, mit der Akteur-Kritiker Struktur und lässt sich auf einzelnen sowie verteilten Systemen einsetzen.[73] Wird mit dem Algorithmus lediglich ein Agent trainiert, wird dies häufig auch als advantage actor-critic (A2C) referenziert.[74] Neben A3C ist der soft actor-critic Algorithmus ein weiterer Algorithmus der Akteur-Kritiker Methodik, welcher die kumulierte Belohnung, aber auch die Entropie maximiert.[75] Durch dieses Konzept der Maximierung der Belohnung und der Entropie ergeben sich die Vorteile eines stärker erkundenden Algorithmus, welcher gleichzeitig mehrere nahezu optimale Lösungen erfassen kann.[76] Zur Evaluation wird die auf der Belohnung und Entropie basierte Wertigkeit iterativ anhand des Bellman Operators nach Bellman 1966 bestimmt, und anschließend die Strategie hinsichtlich der exponentialen Q-Funktion angepasst.[77] Durch die abwechselnde Approximation der Q-Funktion des Kritikers und der Strategie des Akteurs, anstelle der Berechnung dieser bis zur Konvergenz, kann der soft actor-critic Algorithmus besonders mit großen kontinuierlichen Handlungsbereichen umgehen.[78]

2.2.3 Abgrenzung zu Multi-Agent Reinforcement Learning (MARL) Algorithmen

Innerhalb dieses Unterkapitels soll der beschriebene Aufbau von RL Algorithmen und deren Optimierungsproblem zu den von MARL Systemen abgegrenzt werden. Bei MARL Systemen wird anstatt eines Agenten eine Menge von Agenten eingesetzt, welche alle mit ihrer Umgebung interagieren um den Weg der Zielerreichung zu lernen.[79] Dieser Ansatz dient dazu die Vielzahl an Problemstellungen in der echten Welt, welche nicht vollständig durch einen einzelnen Agenten lösbar sind, zu bearbeiten.[80] Einsatzgebiete von MARL sind dabei unter anderem das Routing von Netzwerkpaketen, Wirtschaftsmodellierung oder zusammenhängende Robotersysteme.[81] Je nach Ziel und der demnach definierten Belohnungsfunktion können die Agenten auf die drei unterschiedlichen Arten vollständig kooperativ, vollständig kompetitiv und der Mischung aus beiden miteinander interagieren.[82] Aus den einzelnen Interaktion jedes Agenten mit der selben Umgebung ergibt sich der Unterschied, dass die Umgebungsdynamik aus der Kombination aller Aktionen der Agenten beeinflusst wird anstatt aus der Aktion des einzelnen Agenten.[83] Da dieser Effekt die Annahme der Stationarität von Markov Entscheidungsprozessen verletzt, bedarf die Umgebung einer anderen Representation.[84] Ein Konzept, das dafür häufig verwendet wird,

[72] Vgl. Mnih/Badia u. a. 2016, S. 1
[73] Vgl. Arulkumaran u. a. 2017, S. 9
[74] Vgl. Arulkumaran u. a. 2017, S. 9
[75] Vgl. Haarnoja u. a. 2018, S. 1
[76] Vgl. Haarnoja u. a. 2018, S. 3
[77] Vgl. Haarnoja u. a. 2018, S. 4
[78] Vgl. Haarnoja u. a. 2018, S. 4
[79] Vgl. Wong u. a. 2022, S. 6
[80] Vgl. Canese u. a. 2021, S. 1
[81] Vgl. Canese u. a. 2021, S. 1
[82] Vgl. Canese u. a. 2021, S. 8f.
[83] Vgl. Wong u. a. 2022, S. 2
[84] Vgl. Wong u. a. 2022, S. 6

ist das Markov Spiel, welches sich anders als der Entscheidungsprozess durch einen mehrdimensionalen Aktions- und Belohnungsraum aus der Kombination aller N Agenten auszeichnet.[85] Betrachtet man die Limitierungen von MARL, erkennt man aus den beschriebenen Punkten die Herausforderungen der nicht vorhandenen Stationarität und der Skalierbarkeit, welchen sich die Herausforderung der teilweisen Beobachtbarkeit der Umgebung anschließt.[86]

2.2.4 Limitierungen und Herausforderungen von RL

Trotz signifikanter Errungenschaften birgt der Einsatz der besprochenen RL Algorithmen weiterhin Limitierungen und Risiken für ungewolltes Verhalten.[87]

Eine der Herausforderungen zeigt sich bei der Representation der Agentenumwelt, da RL stark auf diesem Konzept basiert.[88] Daraus ergibt sich die Aufgabe, die Umwelt und deren Verhalten sowie die Wahrnehmung durch den Agenten realitätsgetreu und präzise zu gestalten.[89] Neben der Definition und Wahrnehmung des Umweltverhaltens ist die Spezifikation des Ziels des Agenten ein ebenso kritischer Teil, da unerwartete Intentionen aus der Zielstellung abgeleitet werden könnten.[90] Zusätzlich teilen RL Algorithmen auch Herausforderungen aus anderen Gebieten des maschinellen Lernens wie Genauigkeit, Interpretierbarkeit und die im Rahmen dieser Arbeit untersuchte Robustheit von Modellen.[91]

Eine weitere Limitierung stellt der große Suchraum an Aktionen und das unbekannte Verhalten der Umgebung dar. Dies sorgt dafür, dass die Effizienz einzelner Daten häufig sehr gering ist und die Abwägung zwischen der Exploration neuer Strategie und der Optimierung bekannter Verhaltensmuster ein wichtiger Bestandteil ist.[92] Aufgrund der geringen Effizienz der Daten aber des dennoch hohen Bedarfs an bewerteter Agentenerfahrung wird häufig auf simulierte Daten zurückgegriffen.[93] Simulierte Daten werden dabei häufig von möglichst hoch qualitativen Simulationsumgebungen bereitgestellt, da zu dem hohen Bedarf der Methodik häufig Limitierungen in der Sammlung von Daten in der echten Welt bestehen.[94]

Aufgrund der Bedeutung der Simulationsumgebung und dessen Handlungsraum für RL Algorithmen und deren Transfer in die echte Welt, werden in den beiden nachfolgenden Kapiteln imitierendes Lernen und Simulationen genauer betrachtet.

[85] Vgl. Canese u. a. 2021, S. 4
[86] Vgl. Canese u. a. 2021, S. 9ff.
[87] Vgl. Li 2019, S. 7
[88] Vgl. Sutton/Barto 2018, S. 8
[89] Vgl. Sutton/Barto 2018, S. 7
[90] Vgl. Li 2019, S. 7
[91] Vgl. Li 2019, S. 7
[92] Vgl. Li 2019, S. 7
[93] Vgl. Zhao/Queralta/Westerlund 2020, S. 7
[94] Vgl. Li 2019, S. 8

2.3 Imitierendes Lernen

Für Algorithmen des maschinellen Lernens wie z.B. RL, erweist die Skalierung zu höherdimensionalen Handlungsräumen ein großes Optimierungsproblem auf.[95] Hinzu kommen Schwächen in der Anwendung von verstärkendem Lernen wie Handlungsstrategien in unerfahrenen Umgebungszuständen.[96] Imitierendes Lernen (IL) kann hier als ein Vorschritt zur Optimierung von RL Modellen dienen, womit sich signifikante Belohnungsverbesserungen erzielen lassen.[97] Ein Anwendungsgebiet des Imitierenden Lernens ist die Robotik, in welchem ein chirurgischer Roboter durch Teleoperation bereits erfolgreich chirurgische Eingriffe demonstrieren konnte.[98] Allgemeiner wird IL häufig eingesetzt um menschliches Verhalten nachzuahmen und zu deren Aktionen zu beschleunigen.[99] Das Ziel von IL ist hierbei eine Zuordnung zu erlernen zwischen den Beobachtungen und Aktionen von Expertendemonstrationen.[100] Dadurch soll ein Roboter das ihm demonstrierte Verhalten reproduzieren und auch in unbekannten Szenarien generalisieren können.[101] Problemstellungen können somit auf das Demonstrieren und Anlernen der Fähigkeit beschränkt werden, ohne das explizite Anweisungen oder Belohnungsfunktionen programmiert werden müssen.[102] IL stellt dabei ein passiven Ansatz dar, die Zielstrategie durch Beobachten der vollständigen Ausführungsabläufe zu erlernen.[103] Dabei wird eine Ersatz-Verlustfunktion optimiert, welche die Diskrepanz zwischen der gewählten Aktion der parametrisierten Strategie π_θ und der des Experten ψ misst.[104]

Der typische Einsatzprozess von IL beinhaltet das Akquirieren von Demonstrationsbeispielen, deren Kodierung als Zustands-Aktions Paare und das Optimieren einer Strategie.[105] Einzelne Prozessschritte können sich dabei je nach Anwendungsgebiet unterscheiden, oder Nachfolgeprozesse wie die Feinabstimmung mittels RL enthalten, was an Abbildung Drei deutlich wird.

Algorithmen des imitierenden Lernen unterscheiden sich in der Literatur nach den Begriffen *on-policy* und *off-policy*.[107] Dabei werden zum einen erlernte Strategien in der Umgebung ausgeführt und mittels Experten evaluiert, zum anderen werden lediglich die Expertendemonstrationen verwendet.[108] DAgger stellt ein *on-policy* Algorithmus dar, welcher zu jeder Iteration die Optimierung aller bisher beobachteten Aktions-Zustands Paare vornimmt.[109] Behavioral Cloning als

[95] Vgl. Hussein u. a. 2017, S. 3
[96] Vgl. Attia/Dayan 2018, S. 1
[97] Vgl. Hussein u. a. 2017, S. 4
[98] Vgl. Gao u. a. 2014
[99] Vgl. Attia/Dayan 2018, S. 1
[100] Vgl. Hussein u. a. 2017, S. 1
[101] Vgl. Fang u. a. 2019, S. 365
[102] Vgl. Hussein u. a. 2017, S. 1
[103] Vgl. Attia/Dayan 2018, S. 2
[104] Vgl. Balakrishna u. a. 2020, S. 2f.
[105] Vgl. Hussein u. a. 2017, S. 3
[106] Enthalten in: Hussein u. a. 2017, S. 4
[107] Vgl. Balakrishna u. a. 2020, S. 3
[108] Vgl. Balakrishna u. a. 2020, S. 3
[109] Vgl. Attia/Dayan 2018, S. 5

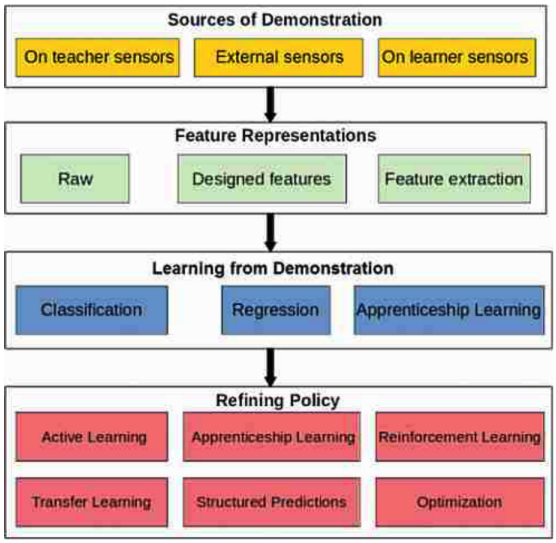


Abb. 3: Ablaufdiagramm des imitierenden Lernens[106]

off-policy Algorithmus erlernt hingegen die direkte Übersetzung der Zustands-und Umgebungsinformationen zu ihren Aktionen, ähnlich einem Label unter überwachtem Lernen.[110]

Insgesamt kann der Einsatz von IL unter anderem folgende Vorteile verzeichnen.[111]

- Verbesserung der Anpassungsfähigkeit hinsichtlich neuer Umgebungen
- Erhöhung der Lerneffizienz aufgrund schnellerer Übertragung von Wissen
- Kompatibilität mit anderen Algorithmen des maschinellen Lernens wie z.B. RL

Aufgrund der interdisziplinären Natur von IL ergeben sich jedoch wie folgt auch eine Reihe von Herausforderungen.[112]

- Der IL Prozess ist sowohl während der Sammlung von Demonstrationsdaten sowie während des Trainingsverlaufs anfällig für Rauschen und Fehler der Sensorik.
- Fähigkeiten, Aufbau und Freiheitsgrade des lernenden Modells und des Experten müssen bestmöglich übereinstimmen.
- Anwendungen lassen sich aufgrund ihres Echtzeitbezugs und Limitierungen der Rechenkapazitäten nur schwierig hinsichtlich hoher Freiheitsgrade skalieren.

[110] Vgl. Fang u. a. 2019, S. 4
[111] Vgl. Fang u. a. 2019, S. 1
[112] Vgl. Hussein u. a. 2017, S. 4f.

2.4 Simulationsumgebungen für RL

Anders als im klassischen Bereich des maschinellen Lernens wie dem überwachten- und unüberwachten Lernen, werden beim verstärkenden Lernen viele der Testdatensätze nicht aus der echten Welt akquiriert.[113] Um entsprechend realistische Daten für das Training bereitzustellen, werden Simulationsumgebungen in Abhängigkeit von ihrer RL Anwendung ausgewählt.[114] Dennoch bleibt nahezu immer eine gewisse Diskrepanz zwischen der Dynamik in der Simulation und der Dynamik in der echten Welt.[115] Möglichkeiten, diese Diskrepanz zu minimieren, sind zum einen das Fehlverhalten von Sensoren einzubinden oder ein reales Signal mit der virtuellen Umgebung zu verknüpfen.[116] Dennoch lässt sich kaum garantieren, dass erlernte Strategien der Agenten sich auf nur leicht veränderte Umgebungen übertragen lassen.[117] Anders als in der Simulation von Flüssigkeiten und deren Dynamik, bedarf RL eine reaktive Umgebung dessen Verhältnis der Simulationszeit zur echten Zeit mindestens eins oder darüber liegt.[118] RL kann von einem erhöhten Echtzeitfaktor profitieren, trotz der damit einhergehenden verringerten Präzision.[119]

2.4.1 Definitionen von Simulationsumgebungen

Ausgehend von der Literaturrecherche zeigt sich, dass in der Forschungsliteratur die allgemeine Definition von Simulationen kaum aufgegriffen wird. Eine mögliche Definition nach Maria 1997 wird wie folgt dargelegt:

Eine Simulation eines existierenden Systems stellt die Anwendung eines Modells dar, welches konfigurierbar zu experimentellen Zwecken das eigentliche System vertritt, um wirtschaftliche oder systematische Herausforderungen des existierenden Systems zu umgehen. Das Model wird in diesem Kontext definiert als Repräsentation des Aufbaus und der Verhaltensweise des existierenden Systems.

Innerhalb bestimmter Anwendungsgebiete, wie der Medizin und der Pflege, werden zusätzlich virtuelle Simulationen wie nachstehend definiert.

Unter virtuellen Simulationen versteht man eine digitale Lernumgebung, welche durch teilweiser Immersion eine wahrnehmbare Erfahrung bereitstellt.[120]

[113] Vgl. Zhang/Wu/Pineau 2018, S. 1
[114] Vgl. Körber u. a. 2021, S. 7
[115] Vgl. Bharadhwaj u. a. 2019, S. 1
[116] Vgl. Zhang/Wu/Pineau 2018, S. 1
[117] Vgl. Bharadhwaj u. a. 2019, S. 1
[118] Vgl. Körber u. a. 2021, S. 3
[119] Vgl. Körber u. a. 2021, S. 3
[120] Vgl. Foronda 2021, S. 1

2.4.2 Entwicklung von Simulationsumgebungen für RL Anwendungen

Im weiteren Teil dieses Kapitels wird aufbauend auf den zuvor angeführten Definitionen, die Entwicklung von Simulationen betrachtet. Allgemein lässt sich dieser Entwicklungsprozess in die folgenden Teilschritte gliedern:[121]

1. Identifikation der Herausforderungen im existierenden System und Ableitung von Anforderungen für die Simulation.
2. Zielgruppe, Funktionsrahmen und quantitative Bewertungskriterien der Simulation definieren.
3. Analyse des zu simulierenden Systemverhaltens durch Sammeln und Verarbeiten von realen Daten des existierenden Systems.
4. Entwicklung einer schematischen Darstellung des Modells und dessen Überführung in nutzbare Software.
5. Validierung des Modells durch bspw. den Vergleich mit dem existierenden System.
6. Dokumentierung des Modells, dessen Variablen, Metriken und getroffene Annahmen.

Die Entwicklung von Simulationen wurde in der Forschungsliteratur besonders durch den Fortschritt im Bereich des verstärkenden Lernens vorangetrieben, da der Vergleich von RL-Algorithmen zuverlässige Benchmarks in Form von Simulationsumgebungen benötigt.[122] Aus dieser Motivation wurde 2016 durch die *OpenAI* der *OpenAI Gym* Werkzeugkasten entwickelt, welcher eine Sammlung an Benchmarksimulationen mit einer einheitlichen Schnittstelle für RL Algorithmen enthält.[123] Seither wurde diese definierte Schnittstelle vielfach verwendet, um RL Umgebungen mit dem Ziel zu entwickeln, diese zu publizieren und dessen Wiederverwendung zu ermöglichen.[124] Die Entwicklung dieses Softwareframeworks wurde nach der Version 0.26.0 im Jahr 2022 durch ein neues Team unter dem Namen Gymnasium weitergeführt.[125] Die Schnittstelle ist definiert als Python Klasse *gym.Env*, von welcher weitere Klassen erben und die vorgeschriebenen Funktionen zum Zeitschritt und zum Zurücksetzen der Simulation implementieren.[126] Der Werkzeugkasten von OpenAI fokussiert sich auf einen „Episoden ähnlichen Rahmen“, in welchem der Agent durch zunächst zufälliges Auswählen von Interaktionen lernt.[127] Weitere Entwicklungsentscheidungen des OpenAI Gym Werkzeugkastens umfassen z.B. die bewusst fehlende Schnittstelle des Agenten, die strikte Versionierung der Umgebung oder die standardmäßige Simulationsüberwachung.[128]

[121] Vgl. Maria 1997, S. 8f.
[122] Vgl. Brockman u. a. 2016, S. 1
[123] Vgl. Brockman u. a. 2016, S. 1
[124] Vgl. Schuderer/Bromuri/van Eekelen 2021, S. 4
[125] Vgl. GitHub 4/4/2023
[126] Vgl. Schuderer/Bromuri/van Eekelen 2021, S. 4
[127] Vgl. Brockman u. a. 2016, S. 1
[128] Vgl. Brockman u. a. 2016, S. 2f.

Werden Lernumgebungen nach der Gym Schnittstelle oder nach eigener Definition für RL Anwendungen eingesetzt, kann sich deren Gestaltung unterschiedlich auf die Leistung der Anwendung auswirken.[129] Eine enge initiale Wahrscheinlichkeitsverteilung des Umgebungszustandes kann die Lerneffizienz erhöhen, wohingegen eine weite Wahrscheinlichkeitsverteilung positiv die Robustheit der erlernten Strategie beeinflusst.[130] Die Robustheit kann zusätzlich durch die Einbindung von Fehlverhalten in der Wahrnehmung der Umgebung beeinflusst werden, da auch in realen Szenarien ein Risiko für Fehlverhalten besteht.[131] Im Bereich der Robotik bzw. in der Simulation von Bewegungen, kann auch durch die Gestaltung des Aktionsraumes, basierend auf elektrischer Regelungstechnik mittels PID-Regler, anstatt basierend auf Drehmomenten ein effizienterer Lernprozess stattfinden.[132]

Neben den beschriebenen Eigenschaften von Umgebungen für verstärkendes Lernen unterliegen auch die verwendeten Simulationen bestimmten Merkmalen, welche in der Entwicklung zu berücksichtigen sind. Laut einer Umfrage nach Ivaldi/Padois/Nori 2014 sind diese wichtigsten Eigenschaften die Stabilität, Geschwindigkeit, Präzision, Genauigkeit, Bedienbarkeit und der Ressourcenverbrauch. Die Entwicklung des Modells, welches das existierende System ersetzt, sollte sich demnach möglichst positiv auf die beschriebenen Eigenschaften auswirken. Neben den beschriebenen Leistungsbezogenen Merkmalen sind die folgenden weiteren Kriterien mitunter die wichtigsten zur Auswahl einer Simulation:

Rank	Most important criteria
1	Simulation very close to reality
2	Open-source
3	Same code for both real and simulated robot
4	Light and fast
5	Customization
6	No interpenetration between bodies

Tab. 2: wichtigsten Kriterien zur Auswahl von Simulatoren[133]

Aus Tabelle zwei lässt sich entnehmen, dass besonders die Nähe zur Realität ein wichtiges Auswahlkriterium ist. Im Kontext von verstärkendem Lernen im Robotik Bereich ist ein wichtiger Baustein die Physik-Engine zur Modellierung von Dynamiken.[134]

2.4.3 Aktuelle Physik-Engines und Simulationsanwendungen

Innerhalb dieses Abschnittes wird aufgrund der Bedeutung der Physik-Engine für den Grad der Simulationsrealität eine Auswahl der aktuellen Physik-Engines und deren Simulationsanwendung betrachtet.

[129] Vgl. Reda/Tao, T./van de Panne 2020, S. 1
[130] Vgl. Reda/Tao, T./van de Panne 2020, S. 3
[131] Vgl. Yan Duan u. a. 2016, S. 2
[132] Vgl. Reda/Tao, T./van de Panne 2020, S. 7
[133] Enthalten in: Ivaldi/Padois/Nori 2014, s. 4
[134] Vgl. Ayala u. a. 2020, S. 2

Gazebo

Gazebo ist eine durch die Open Source Robotics Foundation entwickelte Simulationsanwendung, welche mehrere Physik-Engines unterstützt.[135] Mittels Gazebo lassen sich Interaktionen zwischen Robotern in Innen-und Außenbereichen unter realistischer Sensorik simulieren.[136] Die unterstützten Physik-Engines umfassen Bullet, Dynamic Animation and Robotics Toolkit (DART), Open Dynamics Engine (ODE) and Simbody.[137]

MuJoCo

MuJoCo stellt eine Physik-Engine für modellbasierte Steuerung dar, dessen Objekte durch C++ oder XML definiert und Gelenkzustände im Koordinatensystem beschriebenen werden.[138] Diese beschriebene Eigenschaft lässt sich auch aus dem Namen als Abkürzung für **Mu**lti-**Jo**int dynamics with **Co**ntact ableiten.[139] Die MuJoCo Anwendung ist lizensiert, was den Besitz einer Lizenz für die Installation oder die Virtualisierung innerhalb eines Containers voraussetzt.[140]

PyBullet

PyBullet basiert als Simulationssoftware auf der Bullet-Engine und fokussiert sich funktional auf die Anwendung von RL im Robotikbereich.[141] Die Bullet-Engine ist hingegen eine offene Softwarebibliothek, welche neben verstärkenden Lernen auch bei Computeranimationen angewendet wird.[142] Bie Handhabung von PyBullet profitiert von der ausgiebigen Dokumentation, der großen Entwicklergemeinschaft und der Unterstützung von verschiedenen Dateiformaten wie SDA, URDF und MJCF zur Einbindung von Objekten.[143]

2.5 Simulation der Steuerungsaufgabe von Quadrokoptern

Die Popularität von unbemannten Flugzeugen im letzten Jahrzehnt nahm besonders im Bereich der Quadrokopter zu, sodass durch die sinkenden Kosten von Sensorik und Minicomputern, zahlreiche zukunftsträchtige Ergebnisse und Anwendungen erforscht worden sind.[144] Daher gilt dieses Kapitel der theoretischen Betrachtung dessen Eigenschaften und deren Simulation. Anwendungsgebiete beinhalten z.B. die Landwirtschaft, den Pakettransport oder die Überwachung von großflächiger Infrastruktur wie Stromnetze.[145] Eine Simulation von unbemannten Flugzeugen

[135] Vgl. Ivaldi/Padois/Nori 2014, S. 7
[136] Vgl. Ayala u. a. 2020, S. 4
[137] Vgl. Körber u. a. 2021, S. 3
[138] Vgl. Todorov/Erez/Tassa 2012, S. 1
[139] Vgl. Todorov/Erez/Tassa 2012, S. 2
[140] Vgl. Körber u. a. 2021, S. 3
[141] Vgl. Körber u. a. 2021, S. 3
[142] Vgl. Ivaldi/Padois/Nori 2014, S. 7
[143] Vgl. Körber u. a. 2021, S. 6
[144] Vgl. Koch u. a. 2018, S. 1
[145] Vgl. Deshpande/Kumar, R. u. a. 2020, S. 1

stellt eine Flugumgebung und vielseitige Sensorik bereit und kann je nach Anwendung Effekte wie Wind, Wolken und Niederschlag einbeziehen.[146]

2.5.1 Flugdynamiken eines Quadrokopters

Die Simulation des Quadrokopters stellt eine Simulation eines Flugkörpers mit drei Rotations- und drei Translationsbewegungen und demnach insgesamt sechs verschiedenen Freiheitsgraden dar.[147] Ein Quadrokopter ist ein fester Körper mit vier befestigten Rotoren, welche sich ausschließlich in eine Richtung drehen und positiven Schub in die Z-Achse des Körpers ausüben können.[148] Die vier Rotoren werden als + oder X Konfiguration entweder direkt in Richtung der X- und Y-Achse (+), oder um 45° gedreht (X) an der Drohne befestigt.[149] Jede Bewegung der sechs verschiedenen Arten wird durch die unterschiedliche Ansteuerung, der im Fall des Quadrokopters, vier Rotoren getätigt. Aus den verschieden starken Auftrieben resultieren die drei Rotationsbewegungen Rollen, Nicken und Gieren.[150]

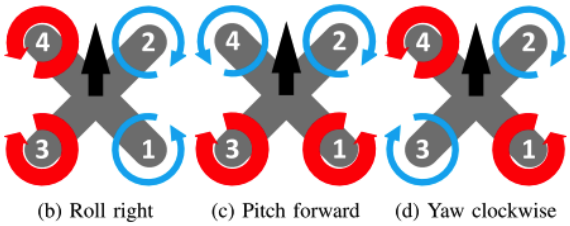


Abb. 4: Rotationsbewegungen eines Quadrokopters[151]

Rollen wird wie aus Abbildung Vier hervorgeht, durch unterschiedlichen Auftrieb der zwei linken und rechten Rotatoren, das Nicken durch Unterschiede der vorderen und hinteren Rotatoren hervorgerufen. Das Gieren bzw. die Rotation um die Z-achse wird durch die stärkere Rotation der sich im Uhrzeiger drehenden oder der sich gegen den Uhrzeiger drehenden Rotoren bewerkstelligt.

Diese Rotationsbewegungen werden mathematisch als Matrix der Eulerschen Winkel repräsentiert, welche die Folge der einzelnen Drehungen entlang der X-, Y-, und Z-Achse enthält.[152] Das Zusammenspiel dieser Rotationsbewegungen, dargestellt anhand Formel vier, mit der durch die Rotatoren entlang der Z-Achse der Drone erzeugte Schubkraft und den Rollraten, beschrieben in Formel fünf, sorgt für die Bewegung der Drohne in Richtung der zukünftigen Koordinaten nach Formel sechs.[153]

[146] Vgl. Hentati u. a. 2018, S. 1496
[147] Vgl. Koch u. a. 2018, S. 2
[148] Vgl. Molchanov u. a. 2019, S. 3
[149] Vgl. Koch u. a. 2018, S. 2
[150] Vgl. Koch u. a. 2018, S. 2
[151] Enthalten in: Koch u. a. 2018, S. 2
[152] Vgl. Deshpande/Kumar, R. u. a. 2020, S. 3
[153] Vgl. Deshpande/Minai/Kumar, M. 2021, S. 2

$$R = \begin{pmatrix} C_\psi C_p & S_\xi S_p C_\psi - S_\psi C_\xi & S_\xi S_\psi + S_p C_\xi C_\psi \\ S_\psi C_p & S_\xi S_\psi S_p + C_\xi C_\psi & -S_\xi C_\psi + S_\psi S_p C_\xi \\ -S_p & S_\xi C_p & C_\xi C_p \end{pmatrix} \quad (4)$$

In Formel vier wird der Rotationszustand der Drohne zu den Weltachsen durch die Sinus- und Cosinuswinkel S_a und C_a repräsentiert, wobei für a die Art der Rotation also Rollen (ξ), Nicken (p) und Gieren (ψ) eingesetzt wird.[154]

$$I \begin{pmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{pmatrix} = \begin{pmatrix} l(F_1 + F_2 - F_3 - F_4) \\ l(-F_1 + F_2 + F_3 - F_4) \\ -M_1 + M_2 - M_3 + M_4 \end{pmatrix} - \begin{pmatrix} p \\ q \\ r \end{pmatrix} \times I \begin{pmatrix} p \\ q \\ r \end{pmatrix} \quad (5)$$

Formel fünf inkludiert in der Matrix I die Trägheitsmomente um die X-, Y- und Z-Achsen in Abhängigkeit der Rollrate p, Nickrate q und Gierrate r.[155] Der Schub der in der Länge l vom Schwerpunkt entfernten Rotatoren wird mittels F_i und das Drehmoment mit $M_i, \forall i \in \{1, 2, 3, 4\}$ gekennzeichnet.[156]

$$m \begin{pmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ -mg \end{pmatrix} + R \begin{pmatrix} 0 \\ 0 \\ \sum_{i=1}^{4} F_i \end{pmatrix} \quad (6)$$

Weiterhin ist die Beschleunigung $\ddot{x}$, $\ddot{y}$ und $\ddot{z}$ in Richtung der drei Achsen durch die Masse m und die Gravitation g beeinflusst.[157]

2.5.2 Existierende Simulationen von Quadrokoptern

RotorS

Die Simulationsumgebung RotorS wurde auf Basis des Robot Operating Systems (ROS) entwickelt, um Programmtestzeiten zu verkürzen, Fehlersuche zu vereinfachen und Unfälle mit echten Mikroflugzeugen zu vermindern.[158] Dabei wurde die Simulation modular entworfen, sodass Komponenten wie Steuerung oder Zustandsschätzung austauschbar sind und das Hinzufügen von neuen Drohnen erleichtert wird.[159] Die Komponenten der Drohne stellen dabei Plug-ins der verwendeten Gazebo Physik-Engine dar, wodurch ein Mikroflugzeug aus den Teilen des Körper, der Anzahl der Rotoren und Sensorik an fixen Position zusammengesetzt wird.[160] Mittels der standardmäßigen Sensorik können Informationen über die direkte und visuell gemessene Trägheit sowie über die Wegbestimmung erzielt werden.[161] Anstelle des Wegbestimmungssensors kann

[154] Vgl. Deshpande/Minai/Kumar, M. 2021, S. 2
[155] Vgl. Deshpande/Minai/Kumar, M. 2021, S. 2
[156] Vgl. Deshpande/Kumar, R. u. a. 2020, S. 3
[157] Vgl. Deshpande/Kumar, R. u. a. 2020, S. 3
[158] Vgl. Furrer u. a. 2016, S. 596
[159] Vgl. Furrer u. a. 2016, S. 595
[160] Vgl. Furrer u. a. 2016, S. 597
[161] Vgl. Furrer u. a. 2016, S. 597

auch eine Komponente zur Zustandsschätzung für hochfrequente Abragen implementiert werden.[162] Die Steuerungskomponente wird durch eine einfache Schnittstelle einer geometrischen Steuerung bedient, welche Aktionen in Form von Rotationswinkeln, Höhen oder Positionen entgegen nimmt.[163]

CrazyS

CrazyS stellt eine Erweiterung der Simulation RotorS auf Basis desselben ROS, um die Modellierung des Nano-Quadrokopters Crazyflie samt ihrer Dynamik ihres Kontrollsystems und ihrer Sensorik dar.[164] Mit der Modellierung der Nanodrohne wurde gleichzeitig ein Konzept zur Erweiterung der RotorS Fähigkeiten dargelegt sowie die Entwicklung von Software-in-the-Loop (SITL) als nahezu Echtzeitüberwachung vorangetrieben.[165]

AirSim

AirSim ist eine open-source Simulationsplattform, mit der das Ziel verfolgt wird, durch eine detailreiche Simulation die Entwicklung von RL und anderen Methoden des maschinellen Lernens voranzutreiben[166] Zur Modellierung der Simulationsphysik wird die Unreal Engine 4 aufgrund ihres hohen Grades an physikalischer und visueller Realität eingesetzt.[167] Der Aufbau der Simulation folgt einem modularen Entwurf, welcher unter anderem die einzelnen Komponenten Fahrzeug, Umgebung, Physik-Engine, Sensorik und Darstellungsschnittstelle beinhaltet.[168] Die Schnittstelle des Fahrzeugs erlaubt eine Steuerung über viele Betätigungselemente und deren Eigenschaftsparameter wie Masse, Trägheit, Widerstand oder Reibung.[169] Ein Fahrzeug ist dabei durch die Umgebungskomponente beeinflusst, welche physikalische Effekte wie Gravitation, Luftwiederstand, Luftdruck und magnetische Felder simuliert.[170] Die Umgebung wird für das Fahrzeug wahrnehmbar durch die Modellierung von Sensoren wie GPS, Beschleunigungsmesser, Gyroskop, Barometer und Magnetometer.[171]

gym-pybullet-drones

Eine weitere nach der Gym Schnittstelle definierte Simulationsumgebung ist gym-pybullet-drones.[172] Basierend auf der Bullet Physik-Engine ermöglicht die Simulation unter anderem das visuell basierte Training von mehreren Agenten mittels RL unter realistischer Modellierung von Kollisionen und aerodynamischen Effekten.[173] Die Wahl der Physik-Engine wurde aufgrund des CPU und

[162] Vgl. Furrer u. a. 2016, S. 598
[163] Vgl. Furrer u. a. 2016, S. 598
[164] Vgl. Silano/Iannelli 2019, S. 81
[165] Vgl. Silano/Iannelli 2019, S. 82
[166] Vgl. Shah u. a. 2017, S. 2
[167] Vgl. Shah u. a. 2017, S. 1
[168] Vgl. Shah u. a. 2017, S. 3
[169] Vgl. Shah u. a. 2017, S. 5
[170] Vgl. Shah u. a. 2017, S. 6
[171] Vgl. Shah u. a. 2017, S. 9
[172] Vgl. Panerati u. a. 2021, S. 1
[173] Vgl. Panerati u. a. 2021, S. 1

GPU basierenden Renderings, des Kollisionsmanagements und der Kompatibilität mit dem Unified Robot Description Format (URDF) getroffen[174] Durch die Kompabilität mit dem URDF Format kann die standardmäßige Simulation des Drohnenmodells Bitcraze Crazyflie 2.x um weitere Nanoquadrokopter erweitert werden.[175]

2.6 Robustheit und Stabilität von Strategien des verstärkenden Lernens

Anders als in Simulationen, lassen sich in der echten Welt häufig Unsicherheiten, Störeinflüsse und grundlegende Veränderungen der Umgebung wahrnehmen, für welche die Methoden des RL standardmäßig nicht robust genug sind.[176] Im nachfolgenden Kapitel wird daher genauer dargestellt, was unter der Robustheit von Algorithmen des verstärkenden Lernens verstanden wird, was Kenngrößen sind und in welchem Kontext diese erfasst werden können.

2.6.1 Definitionen von Robustheit und Stabilität

In der aktuellen Forschungsliteratur findet sich nur wenig Gemeinsamkeit innerhalb der unterschiedlichen Definitionen von Stabilität und Robustheit.[177] Die Definition der Robustheit im Kontext von verstärkendem Lernen wird verschieden interpretiert, wie z.B. als Robustheit gegen Störeinflüsse, Beeinflussung der Belohnung, oder Umgebungsunterschiede.[178] Pullum 2022 definiert Stabilität und Robustheit im Kontext der Literaturanalyse wie folgt:

Stabilität ist eine Eigenschaft des lernenden Algorithmus, die sich auf dessen Leistungsvarianz bezieht und bei geringer Varianz auf ein stabiles Modell hinweist.[179]

Robustheit im Kontext von Software, referenziert eine Eigenschaft eines System, welches nicht nur ausschließlich unter normalen, sondern auch unter außergewöhnlichen Bedingungen, die die Annahmen des Entwicklers übersteigen, gut funktioniert.[180]

Moos u. a. 2022 beschreibt die Robustheit in seiner Literaturanalyse als Fähigkeit mit Variationen und Unsicherheiten in der Umgebung umgehen zu können, wobei Unsicherheiten häufig variierende physische Parameter darstellen.[181]

[174] Vgl. Panerati u. a. 2021, S. 3
[175] Vgl. Panerati u. a. 2021, S. 3
[176] Vgl. Moos u. a. 2022, S. 1
[177] Vgl. Pullum 2022, S. 5
[178] Vgl. Liu u. a. 2023, S. 2
[179] Vgl. Pullum 2022, S. 5
[180] Vgl. Pullum 2022, S. 5
[181] Vgl. Moos u. a. 2022, S. 1

2.6.2 Metriken der Robustheit

Werden Algorithmen und der Erfolg von Veränderungen dieser Experimente betrachtet, kommt es oftmals zu einem Vergleich zwischen Leistung und Stabilität, womit die Belohnung die einzige Kenngröße bildet.[182] Wird diese Metrik im Kontext unterschiedlicher Umgebungen überprüft, kann allerdings so auch die Robustheit von RL Algorithmen betrachtet werden.[183] Neben der Betrachtung der Belohnung lassen sich auch weitere quantitative sowie qualitative Kenngrößen untersuchen, welche die Robustheit und Stabilität eines Algorithmus innerhalb der Umgebung widerspiegeln.[184] In der Literaturanalyse nach Pullum 2022 werden quantitative Metriken zusätzlich nach internen Kenngrößen, welche den Trainingsprozess beschreiben, und externen Kenngrößen, welche die Modellqualität repräsentieren, klassifiziert sowie folgende Tabellen drei und vier zu dessen Metriken angeführt.[185]

Internal Quantitative Metric	**Behavior**	**Total citations**
Reward or Score – magnitude, mean/ variance, variation in average reward, time to threshold, episode duration	Stability, Robustness	75
Policy entropy	Stability	2
Variations in control strategy approximation weights	Stability, Robustness	2
Convergence rate	Stability	2
Lyapunov stability criteria calculated	Stability	1
Policy weight	Robustness	1
Regret	Robustness	1
Wasserstein function bounds calculated	Robustness	1
		85

Tab. 3: Interne quantitative Metriken, dessen gemessenes Verhalten und ihre Häufigkeit[186]

2.6.3 Experimenteller Rahmen zur Messung der Robustheit

Die Tabellen drei und vier zeigen auf, dass trotz der Existenz weiterer Metriken, die Belohnung, deren Durchschnitt, Varianz und Entwicklung am häufigsten eingesetzt werden, um die Robustheit von RL Algorithmen zu messen. Dabei werden Experimente unter festgelegten oder optimierten Hyperparametern, in mehreren Simulationsumgebungen durchgeführt, um aus dem Vergleich derselben Strategie in unterschiedlichen Umfeldern Rückschlüsse auf die Robustheit zu

[182] Vgl. Yan Duan u. a. 2016, S. 6
[183] Vgl. Pinto u. a. 2017, S. 6
[184] Vgl. Pullum 2022, S. 15
[185] Vgl. Pullum 2022, S. 16
[186] Ähnlich enthalten in: Pullum 2022, S.17
[187] Ähnlich enthalten in: Pullum 2022, S.19

External Quantitative Metric	Behavior	Total citations
Deviations/variation in other (than precision, accuracy and recall) performance-related metrics	Stability, Robustness, Resilience	39
Error and failure rates/success rate	Stability, Robustness	28
Performance of tracking/trajectories estimation error; mean absolute deviation, mean square error, mean absolute percentage error, margins and magnitude of correlation coefficient	Stability, Robustness	23
Network-related timing/delay, path and link metrics, connectivity, delivery ratio, routing loops, path optimality, visitation distribution, structural Hamming distance, Small base station-serving ratio, sum-rate and 5th percentile rate	Stability, Robustness	15
Mean/average and variation inaccuracy, precision and recall, area under the receiver operating characteristic (ROC) curve (AUC)	Stability, Robustness, Resilience	12
Variance of the estimation of loss, regret	Robustness	5
		122

Tab. 4: Auszug der externen quantitativen Metriken, dessen gemessenes Verhalten und ihre Häufigkeit[187]

ziehen.[188] Unterschiede in den Umgebungen können bspw. durch fixe Dynamiken wie z.B. Reibwerte während des Trainingsprozesses und unterschiedlicher Reibwerte während der Testphase realisiert werden.[189] Ein weiterer Ansatz kann die Sim-to-Sim Verifikation sein, bei der die optimierten Strategien in einer nicht während des Trainings verwendeten Simulationen untersucht werden.[190]

2.7 Gegnerisches verstärkendes Lernen

Methoden des gegnerischen verstärkenden Lernens verfolgen das Ziel Strategien zu lernen, die robuster gegenüber Risiken wie beeinflusste Wahrnehmung, unbekannte Situationen oder ansteigende Umgebungskomplexität agieren.[191] Die Robustheit gegenüber fehlerhafter Umgebungsbe-

[188] Vgl. Pinto u. a. 2017, S. 5
[189] Vgl. Pinto u. a. 2017, S. 6
[190] Vgl. Molchanov u. a. 2019, S. 5
[191] Vgl. Schott/Hajri/Lamprier 2022, S. 2

trachtung kann durch Störung des Wahrnehmungszustands des trainierenden Agenten erzielt werden.[192] Das Ziel ist es dabei durch gegnerische Aktionen eine veränderte Umgebungswahrnehmnung herzustellen, auf deren Basis sich der lernende Agent verbessert.[193]

Um die lernende Strategie besser auf unbekannte Situationen und steigende Komplexität vorbereiten zu können werden destabilisierende Kräfte in der Dynamik eingeführt.[194] Anders als beim ersten Ansatz werden dafür nicht nur die Wahrnehmung des lernenden Agenten beeinflusst, sondern direkte Einflüsse auf die Umgebung ausgeübt.[195] Hierbei wird der gegnerische Agent dafür belohnt, mittels Kräfteeinfluss die Umgebungsdynamik zu verändern, sodass der lernende Agent an seiner Aufgabe scheitert.[196] Dazu kann ein zusätzlicher Agent mit z.B. gleichem Aktionsraum den gemeinsamen Umgebungszustand beeinrächtigen.[197] Pan u. a. 2021 zeigt eine solche Anwendung im Rahmen der Kontrolle eines Stromnetzes, bei welcher ein gegnerischer Agent für das Trennen von Verbindungen im Netz belohnt wird.[198] Zum Generieren von Störeinflüssen auf die Umgebungsdynamik kann das *Robust Adversarial Reinforcement Learning* (RARL) Framework verwendet werden.[199] Dabei wird der gegnerische Agent selbst durch RL daran angelernt, die möglichst effektivsten Destabilisierungsmaßnahmen zu finden.[200] Formal dargestellt folgt dieses gegnerische Spiel der in Formel sieben angeführten Minimax Optimierung.[201]

$$R^{1*} = \min_{\nu}\max_{\mu} E_{s_0\sim p, a^1\sim\mu(s), a^2\sim\nu(s)}[\sum_{t=0}^{T-1} r^1(s, a^1, a^2)] \quad (7)$$

Zu jedem Zeitschritt t in der gegnerischen Simulation wird von beiden Spielern eine Aktion $a_t^N \sim \mu(s_t) \forall N \in \{1, 2\}$ nach der Wahrnehmung des Umgebungszustands s ausgeübt, was zum Erhalt der Belohnung $r_t^1 = r_t$ und $r_t^2 = -r_t$ führt.[202] Das Training erfolgt aus der abwechselnden Optimierung einer der beiden Strategien bis zu deren Konvergenz, während die jeweils andere nicht verändert wird.[203]

Im Kontext der Steuerung von Quadrokoptern greifen Zhai u. a. 2022 das *Robust Adaptive Ensemble Adversarial Reinforcement Learning Framework (RAEARL)* auf. Unter dessen Einsatz wird der Trainingsprozess des normalen und gegnerischen Agenten zu Gunsten der Kontinuität und Stabilität getrennt, und das System mit einem PID-Regler erweitert, um die Stärke des Gegners über den Trainingsverlauf anzupassen.[204] Das getrennte Training des normalen und gegnerischen Agenten verwendet kopierte Agenten des jeweiligen Gegenparts, welche zwar die

[192] Vgl. Schott/Hajri/Lamprier 2022, S. 2
[193] Vgl. Schott/Hajri/Lamprier 2022, S. 3
[194] Vgl. Pinto u. a. 2017, S. 1
[195] Vgl. Schott/Hajri/Lamprier 2022, S. 2
[196] Vgl. Pinto u. a. 2017, S. 2
[197] Vgl. Pinto u. a. 2017, S. 2
[198] Vgl. Pan u. a. 2021, S. 2
[199] Vgl. Schott/Hajri/Lamprier 2022, S. 2
[200] Vgl. Pinto u. a. 2017, S. 1
[201] Vgl. Pinto u. a. 2017, S. 3
[202] Vgl. Pinto u. a. 2017, S. 3f.
[203] Vgl. Pinto u. a. 2017, S. 4
[204] Vgl. Zhai u. a. 2022, S. 2

selben neuronalen Netzstrukturen und initialen Parameter besitzen, jedoch schwächer sind, da deren Lernfortschritt ein Zeitschritt verschoben ist.[205]

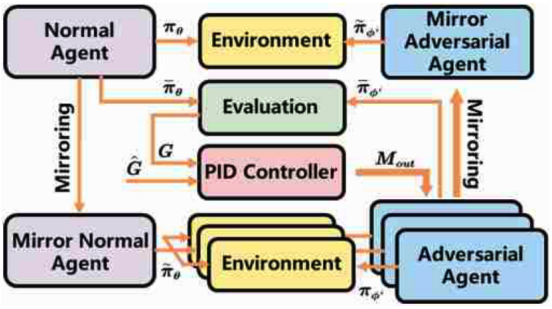


Abb. 5: Aufbau des RAEARL Frameworks[206]

Abbildung Fünf zeigt den beschrieben getrennten Aufbau des RAEARL Frameworks, in dem die Strategien der originalen Agenten mit π_θ bzw. π_{ϕ^i} und jene kopierte Strategien mit $\widetilde{\pi}_\theta$ bzw. $\widetilde{\pi}_{\phi^i}$ notiert sind.[207] Zur Evaluation jeder Epoche und der Bestimmung der kumulierten Belohnung G werden die deterministischen Strategien $\bar{\pi}_\theta$ bzw. $\bar{\pi}_{\phi^i}$ einbezogen.[208]

2.8 Domain Randomization

Domain Randomization (DR) ist eine weitere Methodik die Realitätslücke zwischen der Simulation und der echten Welt zu schließen, in dem die Simulation vielfach variiert wird während des Trainingsprozesses.[209] Im Gegensatz zur Systemidentifikation als Prozess die Parameter bestmöglich an ein physisches System anzupassen, ermöglicht DR einen geringeren Zeitaufwand und geringere Fehleranfälligkeit.[210] Teilweise kann durch diesen Ansatz die Diskrepanz soweit geschlossen werden, dass eine Datensammlung in der echten Welt nicht erforderlich ist.[211] Anhand von Komponenten der Simulationsumgebung lässt sich DR in visuelle und dynamische Randomisierung unterteilen.[212] Weitere Methoden die Realitätslücke zu verkleinern sind progressive Netzwerke, inverse Dynamikmodelle und bayessche Methoden.[213]

2.8.1 Das Prinzip hinter der Randomisierung von Simulationsumgebungen

DR verfolgt den Ansatz eine hohe Anzahl an Simulationsumgebungen mit randomisierten Eigenschaften zu erzeugen, in welchen die Strategie dahingehend optimiert wird, in allen Umgebungen

[205] Vgl. Zhai u. a. 2022, S. 2f.
[206] Enthalten in: Zhai u. a. 2022, S. 3
[207] Vgl. Zhai u. a. 2022, S. 3
[208] Vgl. Zhai u. a. 2022, S.3
[209] Vgl. Bharadhwaj u. a. 2019, S. 3
[210] Vgl. Tobin u. a. 2017, S. 1
[211] Vgl. Molchanov u. a. 2019, S. 2
[212] Vgl. Zhao/Queralta/Westerlund 2020, S. 5
[213] Vgl. Chen u. a. 2021, S. 2

ihr Ziel bestmöglich zu erfüllen.[214] Durch das Aussetzen des Agenten gegenüber einer Vielzahl von Umgebungen, wird unter der Erwartung einer guten Leistung das Ziel verfolgt den Agenten direkt in die physische Welt übertragen zu können.[215] Diese Erwartung resultiert aus der Hypothese, dass unter genug Variabilität die echte Welt lediglich eine bereits trainierte Variation darstellt.[216] Dieses Prinzip kann auch der Abbildung Sechs entnommen werden, in welcher das physische System im Bereich der Randomisierung der Simulation liegt.

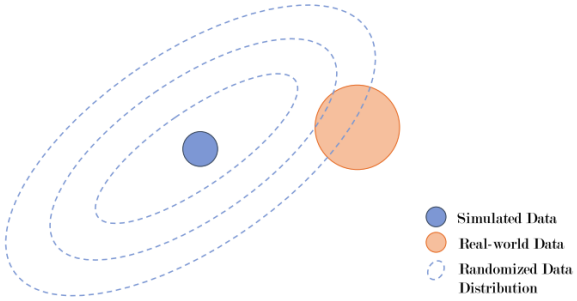


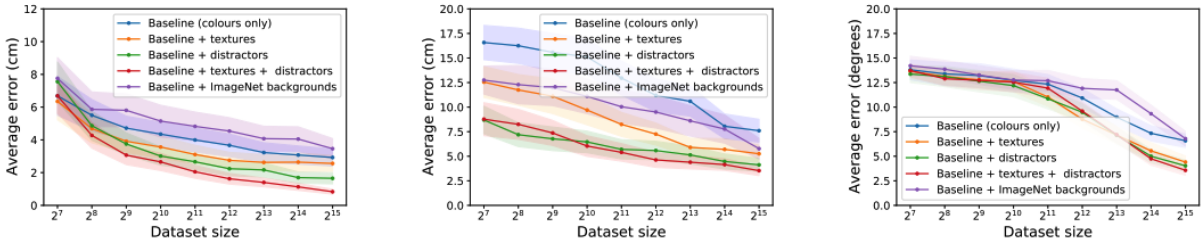
Abb. 6: Intuition hinter dem Paradigma von DR[217]

2.8.2 Anwendung von Randomisierung der Simulationsumgebung

Die Anwendung von DR beginnt nach der Erstellung einer möglichst realen Simulation zum Sicherstellen, dass spätere Variation die wirkliche Welt abbilden können.[218] Anschließend kann die Randomisierung verschiedener Aspekte den Agenten darin unterstützen, die Strategie, welche z.B. als rekurrentes neuronales Netz dargestellt werden kann, so zu optimieren, dass diese für die Wirklichkeit gut genug generalisiert.[219] Aspekte welche im Rahmen von dynamischer DR randomisiert werden können, sind z.B. die Masse von Objekten, die Dauer eines Zeitschrittes oder Schubkoeffizienten von Rotatoren.[220] Verwendet man visuelle Randomisierung so sind bspw. Position, Textur, Ausrichtung und Lichteinwirkung jedes Objektes sowie Kamerawinkel und Zufallsrauschen Eigenschaften, die während des Trainingsprozesses verändert werden können.[221] Alghonaim/Johns 2021 untersuchten dazu detailliert den Einfluss der Randomisierung von Hintergrundfarben und -texturen sowie Ablenkungsobjekte auf die Modellleistung und erhielten folgende Ergebnisse.

Aus der Untersuchung in Abbildung Sieben wird deutlich, dass die Fehleranfälligkeit der Erkennung der X-Y Position, der Z-Position und der Orientierung, durch das Hinzufügen von Texturen und Ablenkungsobjekten mitunter am besten verringert werden kann.

[214] Vgl. Hsu u. a. 2023, S. 1
[215] Vgl. Chen u. a. 2021, S. 2
[216] Vgl. Tobin u. a. 2017, S. 1
[217] Enthalten in: Zhao/Queralta/Westerlund 2020, S. 6
[218] Vgl. Chen u. a. 2021, S. 4
[219] Vgl. Chen u. a. 2021, S. 4
[220] Vgl. Molchanov u. a. 2019, S. 4
[221] Vgl. Tobin u. a. 2017, S. 3
[222] Enthalten in: Alghonaim/Johns 2021, S. 6



(a) xy-position (b) z-position (c) Orientation

Abb. 7: Einfluss der Randomisierung verschiedener Effekte auf die Fehlerrate der Objekterkennung[222]

2.8.3 Errungenschaften unter Einsatz von Domain Randomization

Mit dem Einsatz von DR ist es bereits nach aktuellem Stand der Forschung und Praxis ermöglicht worden, z.B. Strategien ohne reale Bilder in ausschließlich Simulationen mittelmäßiger Qualität zu trainieren und für reale Drohnenflüge im Innenbereich einzusetzen.[223] Weiterhin konnte erreicht werden, ausschließlich aus dem Training in Simulationen eine Objekterkennung und Lokalisation zu einer Genauigkeit von 1,5cm zu realisieren.[224] Neben der Anwendung von DR wird in der aktuellen Literatur sehr stark thematisiert, welche Randomisierung positive Effekte erzielen kann, jedoch wenig zu deren Erklärungen und Funktionsweise detailliert betrachtet.[225] Dies erschwert die Entwicklung effizienter Simulationen und die Bestimmung der Zufallsverteilungen zur Randomisierung.[226]

[223] Vgl. Sadeghi/Levine 2016, S. 1
[224] Vgl. Tobin u. a. 2017, S. 1
[225] Vgl. Zhao/Queralta/Westerlund 2020, S. 6
[226] Vgl. Zhao/Queralta/Westerlund 2020, S. 6

3 Durchführung des Laborexperiments

Anschließend an die Diskussion des aktuellen Stands der Forschung und der Praxis wird im Rahmen dieses Kapitels die Forschungsmethodik angewandt, um die folgende Forschungsfrage zu beantworten:

Inwiefern kann durch den Einsatz eines mittels RL trainierten Gegenspielers die Robustheit der gelernten Strategie verbessert werden?

In der ersten Sektion dieses Kapitels wird auf den Aufbau des Laborexperiments eingegangen. Dabei werden grundlegende Trainingsdaten sowie zu untersuchende Hypothesen, ausgeübte Einflüsse und erfasste Metriken beschrieben. Anschließend werden daraus in der nächsten Sektion resultierende Anforderungen für die Entwicklung der Simulation und der Testumgebung abgeleitet. Die dritte Sektion beinhaltet daraufhin die Umsetzung der beschriebenen Anforderungen und erläutert die endgültige Implementierung der Simulation, der Testdatenerhebung und deren Auswertung.

3.1 Erläuterung der Forschungsmethodik

Innerhalb des Laborexperiments soll die Beziehung zwischen dem Trainingsszenario als Ursache und der Leistungsrobustheit als Wirkung betrachtet werden. Dabei sind zunächst Strategien durch verstärkendes Lernen unter unterschiedlichen Szenarien zu optimieren. Anschließend werden die trainierten Policies in einer veränderten Simulation als Testszenario ausgeübt, und währenddessen verschiedene Metriken der Strategieleistung betrachtet. Ziel ist es, die Leistungsdiskrepanzen zwischen den Strategien des Trainings mit deterministischen und mit optimiertem Gegenspieler zu betrachten. Kein Teil der umzusetzenden Forschungsmethodik ist die Anwendung der simulationsbasierten Strategien zur Steuerung von Quadrokoptern in der echten Welt. Weiterhin wird nicht die Ergebnisabhängigkeit zu Faktoren wie der Wahl des Simulationsframeworks, der Physik-Engine oder des Abstraktionsniveaus untersucht. Die Auswahl jener Aspekte wird unter den Gesichtspunkten der Softwarearchitektur getroffen, dessen Anforderungen und Implementierung in nachfolgenden Sektionen behandelt wird.

3.1.1 Beschreibung der Simulationsumgebung

Beginnend mit dem Training der Policies durch verstärkendes Lernen, wird hierfür anders als bei überwachtem und unüberwachtem Lernen kein unmittelbar vorliegender Datensatz benötigt. Anstelle dessen basiert das Training auf der vollständigen oder teilweisen Wahrnehmung einer Lernumgebung, welche den lernenden Agenten für ausgeführte Aktionen positiv oder negativ belohnt. Wie auch in der Einleitung erwähnt wird im Kontext dieser Arbeit die Simulation von Quadrokoptern dafür verwendet, die Lernumgebung und damit die Trainingsdaten für die

Algorithmen des RL zur Verfügung zu stellen. Die Simulation von Quadrokoptern stellt ein hochdynamisches Anwendungsgebiet dar, bei dem von einer hohen Diskrepanz zwischen der echten und simulierten Welt ausgegangen werden kann. Die Simulationsumgebung stellt grundsätzlich ein Szenario dar, in welchem zwei verschiedene Drohnen kompetitiv gegeneinander agieren. Das Ziel einer Drohne ist es zu einem festgelegten Punkt hinter der zweiten Drohne zu gelangen, währenddessen die zweite Drohne versucht die angreifende Drohne auf seinem Weg abzufangen. Die verteidigende Drohne verwendet dafür das Mittel einer bewussten Kollision. Beide Drohnen werden unter der Einschränkung ihrer Nähe zum Zielpunkt zufällig in einem drei dimensionalem Raum initialisiert, welcher nur horizontal einseitig beschränkt wird. Durch die einseitige Beschränkung des Flugraums wird der natürliche Untergrund dargestellt. Zu Beginn soll die verteidigenden Drohne näher am Zielpunkt als die angreifende Drohne starten, sodass stets ein Abfangen möglich ist. Während der Simulation wird die angreifende Drohne sich entweder regelbasiert auf einer geraden Linie, oder sich durch optimierte Aktionen, zum Zielpunkt bewegen. Die Bewegungsaktionen der verteidigenden Drohne werden durch ein sich optimierendes neuronales Netz bestimmt. Ist das Ziel einer der beiden Drohnen erreicht, oder besteht Kontakt mit dem Untergrund, so wird die Simulation beendet.

3.1.2 Erläuterung der Trainingsszenarien

Durch die erläuterte Simulation werden für das Laborexperiment Strategien in vier unterschiedlichen Szenarien mit einem Algorithmus des verstärkendem Lernen optimiert. Jede der Strategien wird dabei in einer der nachfolgenden Simulationsszenarien optimiert, dessen Auswahl die unabhängige Variable des Laborexperiments darstellt.

1. Das erste Szenario beinhaltet das Training der zu verteidigenden Drohne gegen eine angreifende Drohne, welche eine deterministische regelbasierte Strategie ausführt.

2. Im zweiten Szenario wird eine deterministische Strategie für die anzugreifende Drohne mittels RL optimiert, während die verteidigende Drohne anhand ihrer zuvor optimierten Strategie agiert.

3. Anschließend enthält das dritte Szenario das Training einer Verteidigungsstrategie im kompetitiven Spiel mit der anzugreifenden Drohne unter zuvor optimierter Policy.

4. Zum Abschluss wird das erste Szenario mit regelbasiertem Gegenspieler unter DR, also unter der Randomisierung dynamischer Parameter, als viertes Szenario wiederholt. zur Randomisierung der Trainingsdomäne wird ein konstanter Windeffekt simuliert, dessen Richtung und Stärke zu jeder Trainingsepisode variiert.

3.1.3 Erläuterung des Testszenarien

Nach den Trainingsphasen werden einzelne Policies in mehreren Episoden, eines vom Training abweichenden Testszenarios, ausgeführt und deren Leistungsverhalten gemessen. Eine Episode

entspricht dabei einer Simulation des Testszenarios bis zu ihrem erfolgreichen oder nicht erfolgreichem Ende durch Drohnen-, Ziel-, oder Bodenkontakt. Zusätzlich zu den Trainingsszenarien ist neben den initialen Startpositionen der Drohnen auch der von der Angreiferdrohne zu erreichende Zielpunkt zufällig zu verändern. Neben der zusätzlichen Randomisierung des Zielpunktes wird außerdem ein zum Trainingsszenario verbesserter regelbasierter Gegenspieler verwendet.

3.1.4 Messung der Robustheit von RL Policies

Die in den Testszenarien betrachteten abhängigen Variablen sind die erzielte kumulierte Belohnung, dessen Varianz sowie die Anzahl an unbeabsichtigten Misserfolgen. Die kumulierte Belohnung und dessen Varianz stellen wie bereits beschrieben in der Forschung wichtige Kenngrößen dar, um die Robustheit von RL Policies zu bestimmen. Als Robustheit wird im Rahmen dieser Arbeit die Signifikanz und Stabilität einer Leistungsdiskrepanz von Modellen zwischen Trainings- und Testszenarien definiert. Die Metrik der Anzahl von unbeabsichtigten Misserfolgen spiegelt im behandelten Anwendungsfall das Fehlschlagen der trainierten Strategie wider die gegnerische Drohne nicht abfangen zu können. Mit der Auswahl der Metriken werden die Strategieeigenschaften der maximalen Strategieleistung, der Leistungsabweichung und der Strategiesicherheit deutlich. Aus der Abweichung zwischen dem Leistungsverhalten während des Trainings und während des Tests, soll so die Robustheit von RL Policies erkennbar und messbar gestaltet werden. Der Fokus liegt dabei auf der Messung der Robustheit von den optimierten Strategien aus Szenario eins, drei und vier. Aus dem Vergleich des Leistungsverhaltens zwischen Strategie eins und drei kann eine mögliche Verbesserung der Robustheit gegenüber unbekannten Szenarien abgeleitet werden. Anschließend kann der erzielte Effekt mit der Abweichung zwischen eins und vier verglichen, und so aktuell verwendete Methoden zur Erhöhung der Robustheit einbezogen werden.

3.1.5 Auswertung mittels statistischer Tests

Durch den Vergleich einzelner Strategien mittels dieser Vorgehensweise werden im Laborexperiment die zu Beginn der Arbeit aufgestellten Hypothesen auf ihre Gültigkeit untersucht. Die nachfolgenden bereits zu Beginn angeführten Hypothesen beinhalten wie im vorherigen Kapitel diskutiert, verschiedene Aspekte der Robustheit von RL Strategien, welche in späteren Abschnitten einzeln ausgewertet werden.

1. *Die im Testszenario erzielte kumulierte Belohnung ist unter Verwendung der Policy aus dem Training mit RL basiertem Gegenspieler signifikant und zuverlässig höher als die Policy aus dem Training mit regelbasiertem Gegenspieler.*

2. *Die Varianz der im Testszenario erzielten Belohnung ist unter Verwendung der Policy aus dem Training mit RL basiertem Gegenspieler signifikant und zuverlässig geringer als die Policy aus dem Training mit regelbasiertem Gegenspieler.*

3. *Die im Testszenario erreichte Anzahl von unbeabsichtigten Misserfolgen ist unter Verwendung der Policy aus dem Training mit RL basiertem Gegenspieler signifikant und zuverlässig geringer als die Policy aus dem Training mit regelbasiertem Gegenspieler.*

Jede der drei oben genannten Behauptungen wird in zwei statistischen Tests ausgewertet. Dabei wird zum einen ein Signifikanztest zur Gleichheit der Verteilungen und zum anderen ein Signifikanztest zur Verschlechterung der Metrik durch RL basierten Gegenspieler durchgeführt. Die Thesen zur gleichen oder schlechteren Verteilung der Metrik durch trainierten Gegenspieler werden in den Signifikanztests als H0 Hypothese eingesetzt. Die entsprechenden Gegenhypothesen formulieren jeweils die gegensätzliche Annahme einer Ungleichheit der Verteilungen oder einer Verbesserung der Metrik. Fasst man diesen Aufbau der statistischen Signifikanztests zusammen, können folgende Testhypothesen festgelegt werden.

Gleichheit des ersten Messwerts

- **Hypothese H0:** *Die erzielte kumulierte Belohnung im dritten Testszenario ist signifikant gleich zu der des ersten Testszenarios.*
- **Hypothese H1:** *Die erzielte kumulierte Belohnung im dritten Testszenario ist signifikant unterschiedlich zu der des ersten Testszenarios.*

Verschlechterung des ersten Messwerts

- **Hypothese H0:** *Die erzielte kumulierte Belohnung im dritten Testszenario ist signifikant geringer zu der des ersten Testszenarios.*
- **Hypothese H1:** *Die erzielte kumulierte Belohnung im dritten Testszenario ist signifikant höher zu der des ersten Testszenarios.*

Gleichheit des zweiten Messwerts

- **Hypothese H0:** *Die Varianz der erzielten Belohnung im dritten Testszenario ist signifikant gleich zu der des ersten Testszenarios.*
- **Hypothese H1:** *Die Varianz der erzielten Belohnung im dritten Testszenario ist signifikant unterschiedlich zu der des ersten Testszenarios.*

Verschlechterung des zweiten Messwerts

- **Hypothese H0:** *Die Varianz der erzielten Belohnung im dritten Testszenario ist signifikant höher zu der des ersten Testszenarios.*
- **Hypothese H1:** *Die Varianz der erzielten Belohnung im dritten Testszenario ist signifikant geringer zu der des ersten Testszenarios.*

Gleichheit des dritten Messwerts

- **Hypothese H0:** *Die erreichte Anzahl von unbeabsichtigten Misserfolgen ist im dritten Testszenario ist signifikant gleich zu der des ersten Testszenarios.*

- **Hypothese H1:** *Die erreichte Anzahl von unbeabsichtigten Misserfolgen ist im dritten Testszenario ist signifikant unterschiedlich zu der des ersten Testszenarios.*

Verschlechterung des dritten Messwerts

- **Hypothese H0:** *Die erreichte Anzahl von unbeabsichtigten Misserfolgen ist im dritten Testszenario ist signifikant höher zu der des ersten Testszenarios.*
- **Hypothese H1:** *Die erreichte Anzahl von unbeabsichtigten Misserfolgen ist im dritten Testszenario ist signifikant geringer zu der des ersten Testszenarios.*

Die Auswahl des Signifikanztests wird anhand der Wahrscheinlichkeit einer vorliegenden Normalverteilung vorgenommen. Kann nach einem Kolmogorov- oder Shapiro- Test zu einem Signifikanzniveau von 10% eine Normalverteilung angenommen werden, lässt sich zur Überprüfung der Hypothesen ein T-Test einsetzten. Ist keine Normalverteilung gegeben, wird ein Mann-Whitney U Test verwendet. Liegt schlussendlich der P-Wert des T- oder Mann-Whitney U Signifikanztests zur Prüdung eines Unterschieds unter 10%, so wird die H0 Hypothese abgelehnt und die Abweichung in den Robustheitsdaten als signifikant betrachtet. Zeigt der anschließende statistische Signifikanztest einen P-Wert unter dem Signifikanzniveau, wird H0 abgelehnt und die Verbesserung mittels RL Gegenspieler als signifikant wahrgenommen. Die Beantwortung der Forschungsfrage erfolgt abschließend mit der Betrachtung der angenommenen oder abgelehnten Forschungshypothesen. So werden die verschiedenen Merkmale der RL Policies untersucht und es kann festgestellt werden, welche Effekte auf Robustheit erzielt worden sind.

3.2 Programmanforderungen

Für die Durchführung des Laborexperiments wird ein Softwareprogramm implementiert, welches in der Lage ist, die zuvor beschriebene Methodik umzusetzen. In diesem Kapitel werden die dafür bestehende technische und funktionale Anforderungen genauer thematisiert.

Eine allgemeine Anforderung besteht in der Wahl einer Entwicklungssprache, welche die Entwicklung der Simulation anhand von Softwareframeworks, und zugleich die Integration mit Algorithmen des verstärkenden Lernens erlaubt. Die Entwicklungssprache und verwendete Softwarebibliotheken sollten einer möglichst aktuellen Version entsprechen, und deren Einsatz mit ihrer Version dokumentiert sein. Durch die Dokumentierung kann ein gleiches Abbild der Softwareumgebung auf anderen Instanzen eines Betriebssystems eingerichtet werden.

3.2.1 Anforderungen der Simulationsumgebung

Die übergeordnete Hauptaufgabe der Simulation ist die Generierung von Trainingsdaten für die Optimierung von Strategien mittels RL. Die Qualität der Simulation und besonders der Grad

des Realismus spielen dabei besonders für die spätere Auswertung der Robustheit einzelner Policies eine wichtige Rolle. In diesem Abschnitt wird die Hauptaufgabe in mehrere Anforderungen unterteilt, welche im Entwicklungsprozess umzusetzen sind.

Aus der Betrachtung des aktuellen Stands der Forschung und der Praxis für die Entwicklung von Simulationen geht hervor, dass besonders das OpenAI Gym Framework bzw. dessen Nachfolger Gymnasium in diesem Kontext ein wichtiger Bestandteil ist. Das Gymnasium Framework definiert die wichtigsten Funktionsschnittstellen, welche zur Entwicklung der Simulation im Rahmen dieser Arbeit zu implementieren sind.

Eine der Funktionen beschreibt die **Initialisierung** der eigenen Umgebungsklasse, welche von der Umgebungsklasse der Gymnasium-Bibliothek erbt. Initial sind darin für die Simulation die Startpositionen der verteidigenden Drohne und der angreifenden Drohne sowie der anzugreifende Zielpunkt anzugeben. Die Simulation soll eine zufällige Platzierung der Drohnen und des Zielpunktes im Raum ermöglichen, jedoch ausschließlich unter der Bedingung, dass die direkte Strecke zum Zielpunkt für die verteidigende Drohne kürzer ist, als für die angreifende Drohne. Zur Erfüllung dieser Anforderung muss eine Wahrscheinlichkeitsfunktion für die Zielpunktkoordinaten sowie je Drohne, für die drei Startkoordinaten X,Y und Z bestimmt werden. Die Randomisierung dieser Koordinaten soll über eine Variable an- und abgeschaltet werden können, sodass einfach zwischen Trainings- und Testszenario gewechselt werden kann. Als Spielgebiet wird eine flache feste Ebene auf der Höhe Null eingesetzt, über dessen der drei dimensionale Raum unbegrenzt in X-, Y-, und Z-Richtung zur Verfügung steht.

Eine weitere Funktionen beinhaltet die Definition des **Beobachtungsraums**, also welche Informationen für den Agenten Wahrnehmbar sind. In der beschriebenen Simulation soll der Agent die Position, die Ausrichtung, die Richtungsgeschwindigkeit der Drohne und die Geschwindigkeit der Rotatoren wahrnehmen. Jede diese Eigenschaften wird von der verteidigenden Drohne und zusätzlich von der angreifenden Drohne wahrgenommen. Insgesamt stelle der Beobachtungsraum somit ein Datenobjekt dar, welches alle Informationen wie Koordinaten, Geschwindigkeiten und Drehzahlen beinhalte.

Neben dem Beobachtungsraum muss auch ein **Handlungsraum** festgelegt werden, welcher die Steuerung der verteidigenden Drohne umfassen soll. Hierbei soll durch den Agenten die Geschwindigkeit und ihre Richtung in Form eines vier-elementigen Liste vorgegeben werden. Auch die regelbasiert gesteuerten Drohnen folgen damit dem Ziel eine dieser Form entsprechende Aktion zu erzeugen. Anschließend besteht die Anforderung diese Aktionen in einen konkreten Einfuss auf den Zustand des Quadrokopters zu übersetzen.

Führt der Agent eine Aktion seines Handlungsraums aus, wird die Implementierung eines **Zeitschritts** der Simulation benötigt. In dieser muss deklariert sein, wie sich die gewählte Aktion auf den Zustand der Drohnen auswirkt. Während jedes Zeitschritts der Simulation sind dabei die Kriterien zum Zurücksetzen der Simulation zu prüfen.

Im Anwendungsbereich dieser Arbeit ist die Simulation zurücksetzen, sobald das Abfangen der angreifenden Drohne erfolgt ist, oder eine der Drohnen in Kontakt mit dem Untergrund kommt. Das Gymnasium Framework definiert die **Rücksetzfunktion**, welche den initialen Simulationszustand herstellt, sobald ein Kriterium erfüllt ist. Durch die Schrittfunktion ist abschließend die neue Wahrnehmung der Simulation, die Erfüllung des Rücksetzkriteriums und der Belohnungswert zurückzugeben.

Zur Berechnung einer entsprechenden Belohnung, für die Ausführung der gewählten Aktion, ist eine **Belohnungsfunktion** einzusetzen. Diese muss anhand des neuen Simulationszustandes Kennzahlen ermitteln und gewichten, und dadurch eine numerische Bewertung der neuen Situation vornehmen. Hinsichtlich der Maximierung des Belohnungswertes, optimiert der Agent mittels des RL Algorithmus seine Aktion in Abhängigkeit der Simulation. Die Anforderungen dieser Optimierung der Aktionen werden in der nachfolgenden Sektion detailliert betrachtet.

3.2.2 Anforderungen des Optimierungsverfahrens

Das Optimierungsverfahren trägt die Aufgabe zur Bestimmung der bestmöglichen Aktion zu jedem Zeitschritt der Simulation. Dabei ist die Strategie zur Steuerung des Quadrokopters mittels RL-Algorithmus zu optimieren. Da die eigene Implementierung des RL-Algorithmus nicht im Fokus dieser Arbeit liegt, werden frei verfügbare Softwarebibliotheken verwendet. Eine Bibliothek welche eine Vielzahl an RL-Algorithmen beinhaltet ist Stable-Baselines3 nach Raffin u. a. 2021. Die Softwarebibliothek integriert weitere Bibliotheken wie Tensorboard zum Messen von Trainings- und Evalutationskennzahlen, was die Durchführung der Forschungsmethodik unterstützt. Weiterhin ist eine umfangreiche Dokumentation zur Implementierung von RL Algorithmen in Verbindung mit dem Gymnasium Framework vorhanden, was die Umsetzung des Trainingsprozesses in Verbindung mit der Simulationsumgebung erleichtert. Aus der Softwarebibliothek Stable-Baselines3 soll der PPO-Algorithmus zum Optimieren je einer Policy in jedem Trainingsszenario eingesetzt werden.

Der Trainingsprozess beinhaltet im ersten Schritt die Bestimmung der unabhängigen Variablen. In Kontext unserer Arbeit ist dies die Wahl des Trainingsszenarios, dessen Eigenschaften in der Simulation einzustellen sind. Alle Eigenschaften der Szenarios sollen über die Veränderung weniger Parameter als zentrale Schnittstelle im Programm auswählbar sein. Anschließend sind die gewählten Metriken anzugeben, welche im Trainingsprozess erfasst werden sollen. Diese beinhalten die durchschnittliche Episodenlänge und kumulierte Belohnung sowie die Erfolgsrate des Zusammenstoßes beider Drohnen, über alle im Trainingsprozess evaluierten Strategien. Zur Erfassung dieser Trainingsmetriken gilt es, die Integration mit der Tensorboard Bibliothek zu nutzen. Daraufhin ist das Training über die dafür vorgesehene Funktion des RL-Modells zu starten und nach dessen Ablauf das optimierte Modell zu speichern. Das Speicherformat ist so zu wählen, dass gespeicherte Modelle zur Auswertung in das Testszenario geladen werden können.

Aufgrund des hohen dreidimensionalen Aktionsraums, welcher zu jedem Zeitschritt zu optimieren ist, soll vor dem beschriebenen Trainingsprozess ein Training durch IL eingesetzt werden. Im IL Prozess ist ein Modell dahingehend zu optimieren, eine vorgeschriebene regel-basierte Strategie zur Steuerung des Quadrokopters nachzuahmen. Die Policies der später mittels RL zu optimierenden Quadrokopter werden demnach zunächst regelbasiert als Experte definiert. Daraufhin ist es das Ziel die Gewichte eines IL-Modells wie z.B. Behavioral Cloning so anzupassen, dass diese bestmöglich die regelbasierte Strategie wiederspiegeln. Anschließend sind die in diesem Prozess erlernten Gewichte zu einem RL-Modell zu transferieren und mittels RL-Algorithmus zu verfeinern. Durch diesen Transfer soll eine hohe Qualität der RL-Modelle gewährleistet werden, sodass die Messung der Robustheit valide Ergebnisse erzielt.

3.2.3 Anforderungen des Laborexperiments

Anhand der Implementierung des Laborexperiments sollen die notwendigen Messdaten erhoben, die Forschungshypothesen evaluiert, und die Forschungsfrage beantwortet werden. Dazu ist es notwendig die beschriebenen Metriken, also die kumulierte Belohnung, dessen Varianz und die Anzahl an Misserfolgen, zu allen Policies im Testszenario zu erfassen. Das Testszenario wird durch eine Instanz der Simulationsumgebung repräsentiert, die eine zum Training veränderte Situation beinhaltet.

Die Abweichung der Domäne zu den Trainingsszenarien wird durch die zusätzliche Randomisierung des Zielpunktes und durch die Verbesserung des regelbasierten Gegenspielers erzielt. Die Verteilung des Zielpunktes soll dabei ähnlich zu den Startkoordinaten so gewählt sein, das sich die verteidigende Drohne näher an ihr befindet. Das Abfangen eines direkten Anflugs der anzugreifenden Drohne bleibt dadurch möglich. Im Testszenario soll die Verbesserung der anzugreifenden Drohne aus einer parabelförmigen anstatt geraden Fluglinie zum Zielpunkt bestehen. Damit die Validität der Messdaten gegeben ist, muss sichergestellt sein, dass alle Strategien den gleichen Testbedingungen und demnach den selben zufälligen Start- und Zielpunkten unterliegen. Hierfür ist ein Zufallswert zu übergeben worunter bei jedem Programmstart die gleichen zufälligen Testbedingungen hergestellt werden. Anschließend ist jede Policy in mehrfachen verschiedenen Episoden auszuführen, und deren Leistungsverhalten zu messen.

Innerhalb eines Testszenarios sind zu jedem Zeitschritt die aktuelle Belohnung zu speichern und zu Messen, ob ein Misserfolg der Drohne vorliegt. Beide Variablen folgen dem Format einer Zeitreihe über den Verlauf der Testsimulation. Das Programm zur Auswertung muss nach 100-facher Ausführung der Policy im Testszenario die Kennzahlen der durchschnittlichen Belohnung und dessen Varianz berechnen. Zur Dokumentation und für die weitere Auswertung mittels statistischer Tests sind die Messdaten zu speichern.

Durch das Laden der Messdaten aus Dateien in entsprechende Datenobjekte liegen alle Metriken zur Bestimmung des statistischen Tests vor. Anschließend sind die Daten durch das Auswertungsprogramm auf ihre Verteilung hin zu untersuchen. Hierfür ist ein Kolmogorov- oder

Shapiro- Test, anhand verfügbarer Softwarebibliotheken zu verwenden. Können die Messdaten als normalverteilt angenommen werden, gilt es ein T-Test, andernfalls einen Mann-Whitney U Test zur Überprüfung der Hypothesen einzusetzen.

Die genaue Implementierung dieser Tests soll ebenfalls durch zusätzliche Softwarebibliotheken bestimmt sein. Anhand der Testergebnisse sind durch das Programm die H0 und H1 Hypothesen zu evaluieren und darauf aufbauend die Forschungshypothesen zu bestätigen oder zu verwerfen. Zusätzlich zu den Messdaten der Testszenarien sind auch die Ergebnisse der Signifikanztests zu speichern. Schlussendlich sind in einer Tabelle zu jeder Forschungshypothese die Ergebnisse der statistischen Tests anzuführen. Durch die gesamte einheitliche Betrachtung der Robustheitsmerkmale kann final die Forschungsfrage beantwortet werden.

3.3 Implementierung der Simulation und des Experiments

Fortfolgend an die in der letzten Sektion beschriebenen Anforderungen, wird in dieser Sektion die finale Implementierung der einzelnen Programmabschnitte erläutert. In Anlehnung an den Aufbau der letzten Sektion, wird die Umsetzung der Anforderungen zu den Programmabschnitten Simulationsumgebung, Trainingsverfahrens und Laborexperiment dargelegt.

Allgemeiner Natur ist die Wahl der Entwicklungssprache Python. Die Entwicklungssprache ermöglicht eine hohe Kompatibilität mit bekannten Simulationsframeworks, RL-Softwarebibliotheken und Bibliotheken zur statischen Auswertung von Messdaten. Im Rahmen dieser Arbeit wird konkret die Python Version 3.10 eingesetzt, um von den Möglichkeiten neuer Funktionen zu profitieren und zugleich eine möglichst hohe Kompatibilität zu existieren frei verfügbaren Bibliotheken aufzuweisen. Alle in dieser Arbeit verwendeten Bibliotheken sowie deren Abhängigkeiten sind in einer virtuellen Umgebung installiert, sodass bestmöglich Störeinflüsse durch bereits installierte Pakete vermieden werden. Ebenso sind alle eingesetzten Softwarepakete in einer Textdatei inklusive ihrer Version dokumentiert.

3.3.1 Programmumsetzung der Simulationsumgebung

Die Umsetzung der Anforderungen der Simulationsumgebung beinhaltet im Fokus die Entwicklung einer Simulationsumgebung anhand des Gymnasium Programmiergerüsts. Die entwickelte Simulation baut dabei auf bestehende Programmbibliotheken auf. Dies fördert vor allem die im Zuge dieser Arbeit umsetzbare Qualität und ermöglicht einen hohen Grad an Realismus, welcher wie in den Anforderungen beschrieben unmittelbar das Sim2Real Problem und damit die Robustheit beeinflusst. Weiterhin stellt die Neuentwicklung einer Simulationsumgebung kein Entwicklungsaufwand dar, welcher einen Einfluss auf die auszuwertende Forschungsfrage ausübt. Wird die im zweiten Kapitel angeführte Auswahl an in der Literatur beschriebenen Simulationsumgebungen betrachtet, kann aus diesen eine Basis zur Entwicklung der eigenen Simulationen gewählt werden.

Zur Auswahl einer Basissimulation, in welcher die eigenen Trainings- und Testszenarien abgebildet werden können, sind unterschiedliche Kriterien zu beachten. Ein Kriterium ist die Kompatibilität der Basissimulation mit der gewählten Entwicklungssprache. Hierbei sollte entweder die Simulation selbst in Python, oder eine entsprechende Schnittstelle entwickelt sein. Die gewählte Simulation sollte eine Physik-Engine einsetzen, welche zum einen einen möglichst hohen Grad an Realismus erlaubt, um die Forschungsfrage möglichst valide zu beantworten. Zum Anderen sollte die Simulation einen beherrschbaren Rechenleistungsaufwand verursachen, da die Zielsetzung die Optimierung mehrerer Policies beinhaltet. Weiterhin, zur Erfüllung der zuvor beschriebenen Anforderungen, muss die Basissimulation dem Framework Gym oder dessen Nachfolger Gymnasium entsprechen. Um die Entwicklung des Optimierungsverfahrens zu unterstützen wird eine RL Integration oder zumindest eine Umfangreiche Dokumentation zu dessen Einsatz vorausgesetzt. Die nachfolgende Tabelle stellt die in Betracht gezogenen Drohnensimulationen den für die Auswahl getroffenen Kriterien gegenüber.

	RotorS	CrazyS	AirSim	gym-pybullet-drones
Python API	X	X	X	X
Integration des Gymnasium Frameworks	X	X	X	X
höchster Realismusgrad			X	
kontrollierbarer Rechenleistungsaufwand	X	X		X
umfangreiche RL Integration und Dokumentation			X	X
Simulation mehrerer Quadrokopter enthalten				X

Tab. 5: Gegenüberstellung von Auswahlkriterien und bekannten Drohnensimulationen

Aus der Gegenüberstellung von Auswahlkriterien und aktuellen Simulation geht hervor, dass zunächst alle Simulationen eine Schnittstelle für die Entwicklungssprache bereitstellen. Dabei wurden native Schnittstellen sowie Schnittstellen aus zusätzlichen Softwarebibliotheken miteinbezogen. Auch die Integration des Gym oder Gymnasium Frameworks wird durch alle Simulationen eigens oder durch zusätzliche Bibliotheken sichergestellt. Bei der Betrachtung der Realitätsnähe zeigt die AirSim Umgebung aufgrund der verwendeten Unreal Physik-Engine den höchsten Grad an Realismus auf. Im Gegenzug werden durch diese Simulation Rechenkapazitäten erwartet, welche im Rahmen dieser Arbeit nicht zur Verfügung stehen. Eine umfangreiche Dokumentation und bereits vorhandene Integration von RL wird durch AirSim und gym-pybullet-drones gewährleistet. Die Simulationen RotorS und CrazyS bieten eine Integration mit RL nur auf Basis der wenig dokumentierten zusätzlichen Softwarebibliotheken GymFC und gym_multirotor. Wird das letzte Kriterium der Simulation mehrerer Drohnen betrachtet, tritt dies nur in Beispielen der gym-pybullet-drones Simulation auf. Fasst man die Erfüllung der Auswahlkriterien

[226] Eigene Darstellung

zusammen wird erkenntlich, dass gym-pybullet-drones als einzige Drohnensimulationen die zusätzlichen Anforderungen der RL Integration und der Dokumentation erfüllt. Demnach wird für die nachfolgende Entwicklung der eigenen Simulation auf der gym-pybullet-drones Bibliothek aufgebaut.

Die gym-pybullet-drones Simulation ist in einzelne Simulationszenarien gegliedert, welche je eine Python Datei umfassen. Als Aviaries gekennzeichnet, bilden sie je eine Trainingsumgebung nach der Schnittstellendefinition des Gymnasium Frameworks ab. In der Struktur der Simulationsumgebung wird das Programmierkonzept der Vererbung verwendet, um ähnliche Simulationen auf den selben übergeordneten Klassen zu basieren. Jede erbende Klasse enthält damit alle Funktionen und Variablen der übergeordneten Elternklasse. Dies ermöglicht die verschieden starke Abstraktion der Simulation auf den unterschiedlichen Vererbungsebenen. Weiterhin ermöglicht die Bibliothek verschiedene Steuerungsarten der Quadrokopter. Eine Option ist die Steuerung über die direkte Vorgabe der Drehzahlen aller Rotatoren. Zusätzlich kann auch ein PID-Kontroller eingesetzt werden um die Steuerungssignale entgegenzunehmen. Zur Entwicklung der eigenen Simulation wird die Steuerung über einen Richtungsvektor und einen Geschwindigkeitswert gewählt. Unabhängig des gewählten Aktionstyps werden alle Steuerungssignale durch Controllerklassen in konkrete Drehzahlen übersetzt. Dessen genaue Übersetzung wird im Rahmen dieses Kapitels nicht erläutert und ist in der entsprechenden Dokumentation nach Panerati u. a. 2021 nachzulesen. Auch der Beobachtungsraum kann zum einen visuell und zum anderen kinetisch erfolgen, was im Rahmen dieser Arbeit verwendet wird. Der kinetische Beobachtungsraum stellt eine Reihe von wahrnehmbaren Eigenschaften wie Position, Ausrichtung, Geschwindigkeit und Drehzahl dar. Unabhängig der Überwachungsart kann durch die pybullet Physik-Engine ein grafisches Modell der Simulation erzeugt werden welches in Abbildung Acht dargestellt wird.

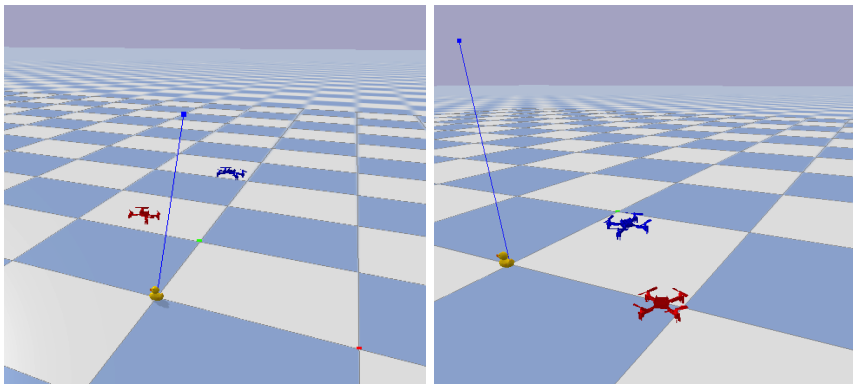


Abb. 8: grafisches Modell der Simulation[227]

Die Umsetzung, der zu Beginn dieses Kapitels beschriebenen Simulationsumgebung, basiert auf der Verwendung der Basissimulation und der Kontrollerklassen. Zusätzlich sind teilweise Funktionen aus ähnlichen bereits nativ vorhandenen Szenarien mit Änderungen übernommen und im Programm entsprechend gekennzeichnet. Die beschriebene Simulation ist innerhalb einer Klasse nach dem Gymnasium Framework aufgebaut. Zusätzlich wird die Simulation durch eine erbende

[227] Eigene Darstellung

Klasse umhüllt, um für den Algorithmus eine verarbeitbare Abstraktionsschicht zu erzeugen. In dieser Schicht wird die Steuerung der angreifenden Drohne abstrahiert, sodass die Trainingsumgebung lediglich die Bestimmung einer Aktion für die verteidigende Drohne erwartet. Nachfolgend wird die Umsetzung dieser beiden Klassen nach den einzelnen Funktionen des Gymnasium Frameworks erläutert.

Die Simulation beider Quadrokopter ist in der Klasse **DualDroneAviary** definiert. Zur Initialisierung einer Simulationsinstanz wird die **Init-Funktion** aufgerufen, welche die Übergabe aller simulations relevanten Parameter erwartet. Die Parameter umfassen unter anderem Informationen zur Anzahl und Art und Position der Drohnen, gewählte Handlungs- und Beobachtungsarten sowie die unabhängigen Variablen der Trainingsszenarien. Außerdem wird die Gewichtung der Belohnungsfunktion als Parameter entgegen genommen. Kernaufgabe dieser Funktion ist die Umsetzung der Eigenschaften eines Trainings- und Testszenarios. Dies beinhaltet die Erzeugung und Übermittlung der zufälligen Start- und Zielpunkte und des zufälligen Windeffektes mittels unterstützender Funktionen aus der eigenen Hilfsklasse. Die verteidigende Drohne wird zufällig innerhalb von 1.5 Meter in X- und Y-Richtung und 0.1 bis 3 Meter in Z-Richtung zum Ursprung initialisiert. Hingegen die angreifende Drohne wird im Rahmen von zwei bis sechs Metern uniform um den Ursprung herum auf einer möglichen Höhe von 0.1 bis fünf Meter platziert. Sind die Eigenschaften bestimmt, können diese zur Erzeugung der darunter liegenden Abstraktionsschicht der BaseAviary Instanz genutzt werden.

Der **Beobachtungsraum** in dieser Simulation wird durch einen Informationsvektor definiert. Der für die Experimente eingesetzte Informationsvektor je Drohne enthält die Position, Ausrichtung, Fortbewegungs- und Drehgeschwindigkeit sowie die Rotationsdrehzahlen. Die Position ist durch die dreidimensionale Koordinaten, die Ausrichtung durch Quaternion und Drehwinkel kodiert. Positionen können dabei von $-\inf$ bis $+\inf$ in X- und Y-Richtung und von 0 bis $+\inf$ in Z-Richtung betrachtet werden. Die Quaternion erlauben Werte zwischen -1 und $+1$, die Drehwinkel von $-\pi$ bis $+\pi$ Drohnen-, Dreh-, und Rotatorengeschwindigkeit werden als Geschwindigkeitsvektor von minus bis plus unendlich zu den Koordinatenachsen bzw. als vier einzelne Werte von Null bis zu maximalen Drehzahl definiert. Insgesamt ist für jede Drohne ein Beobachtungsvektor mit 20 Elementen innerhalb eines Wörterbuchobjektes definiert.

Die Definition des **Aktionsraums** ist in der ActionSpace Funktion bestimmt. Je nach Aktionstyp ist ein drei oder vierdimensionaler Vektorbereich definiert. Jeder Wert des Vektors kann besitzt eine untere und obere Grenze aus dessen Bereich die endgültige Aktion gewählt wird. Zur Steuerung des Quadrokopters wird im Rahmen dieser Arbeit der Aktionstyp auf Basis der Geschwindigkeit verwendet. Der Wertebereich des Richtungsvektors ist dabei von $[-1, -1, -1]$ bis $[1, 1, 1]$ definiert. Das vierte Element spiegelt den Anteil der maximalen Geschwindigkeit von null bis eins wieder.

Ist eine Aktion regelbasiert oder durch einen RL-Modell bestimmt, wird diese als Parameter an die **Step-Funktion** weitergegeben. Auf der Ebene der DualDroneAviary Klasse kann die übergebene Aktion in Abhängigkeit vom anfangs initialisierten Wind abgewandelt werden. Der

Windvektor wird in Abhängigkeit des Winkels zwischen Bewegungsrichtung der Drohne und des Windes zur Aktion hinzugefügt. Der Drohnenbewegungsrichtung wird die Windrichtung addiert und die Geschwindigkeit in Abhängigkeit des Eintreffwinkels vollständig postiv, negativ oder gar nicht beeinflusst. Anschließend wird unter veränderter oder unveränderter Aktion die Schrittfunktion der Elternklasse aufgerufen. Die Schrittfunktion der Elternklasse übersetzt die Aktion aus dem Aktionsraum in konkrete Rotorendrehzahlen und übt diese unter der simulierten Physik aus. Zur Erhöhung des Realismusgrads wird zusätzlich zu den Effekten der pybullet Physik-Engine ein Boden- und Trägheitseffekt eingesetzt. Anschließend wird die neue Zustandsbeobachtung, die entstandene Belohnung, das Rücksetzkriterium und ein Datenobjekt für Zusatzinformationen zurückgegeben.

Die Bestimmung des Rewards ist unter der **reward-Funktion** der DualDroneAviary Klasse implementiert. Innerhalb dieser Funktion werden aus dem aktuellen Zustand der Simulation Kennzahlen zu dessen Evaluation berechnet und gewichtet. Durch die Gewichtung und dessen Optimierung kann das Strategieziel formuliert und der Einfluss des Wertebereichs der Kennzahlen angepasst werden. Die Aufsummierung der Produkte aus den folgenden Kennzahlen und deren Gewichtung spiegelt die Belohnung zu jedem Zeitschritt wieder.

- Distanz zwischen der angreifenden und verteidigenden Drohne
- Belohnung bei erfolgreichem Abfangen der angreifenden Drohne
- negative Belohnung sofern der Zielpunkt erreicht wurde
- Belohnung für eine möglichst direkte Flugrichtung zur gegnerischen Drohne
- Lineare Belohnung der Geschwindigkeit

Eine Optimierung der Gewichte wurde durch Ausprobieren unterschiedlicher Gewichtskombinationen und Messen der entsprechenden Erfolgsrate erzielt. Zu jedem Zeitschritt eintretende Kennzahlen wurden aufgrund ihrer Häufigkeit in Fünfer Schritten, außerordentliche Belohnungen in 10000er Schritten untersucht. Insgesamt sind daraus unter anderem folgende in Abbildung Neun dargestellte Log-Dateien entstanden, welche den Optimierungsprozess wiederspiegeln.

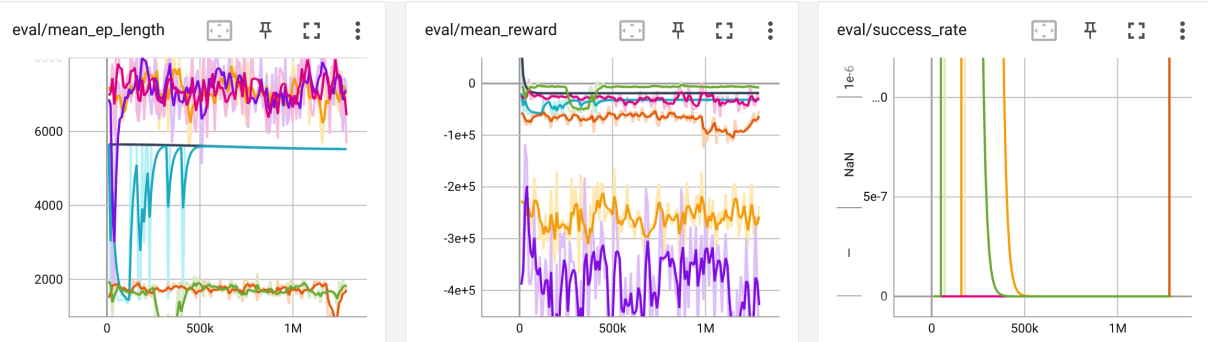


Abb. 9: Episodenlänge, Belohnung und Erfolgsrate unter verschiedenen Gewichtungen[228]

[228]Diagramme durch die Tensorboard Bibliothek erstellt

Die Optimierung der Belohnungsfunktion brachte $[-10, 50000, 50000, 25, 20]$ als optimalen Gewichtsvektor hervor, da hierunter mehrere Algorithmen wie PPO und A2C, eine positive Erfolgsrate verzeichneten.

Ist das Rücksetzkriterium erfüllt wird die Simulationsumgebung mittels **reset-Funktion** neugestartet. Dabei sind neue Zufallspositionen der Drohnen und des Ziels sowie im vierten Trainingsszenario eine neue Windeigenschaften zu generieren. Sind diese Variablen verändert kann die Rücksetzfunktion der darunterliegenden Abstraktionsschicht aufgerufen werden. Die Funktion der BaseAviary-Klasse setzt darin die Zeitschritte und die Drohneninformationen zurück und baut das grafische Modell neu auf. Über den Zugriff auf den Klienten der pybullet Physik-Engine werden die Drohnenobjekte sowie das Untergrundobjekt aus den URDF-Dateien geladen und an entsprechenden Stellen platziert. Zur Unterscheidung des verteidigenden Quadrokopters von der angreifenden Drohne werden zwei nahezu identische URDF-Dateien verwendet, welche sich nur in der Farbgebung zwischen rot und blau unterscheiden. Am Ende der Funktion wird der erste Beobachtungsvektor der neuen Episode sowie ein Infoobjekt zurückgegeben.

Die zuvor beschriebene Klasse vererbt ihre Variablen und Funktionen auf die höhergelegene Abstraktionsschicht der **SingleDroneWrapper** Klasse. Auf dieser Ebene wird bietet die Simulation ihre Schnittstelle für die Verwendung durch den IL- und RL-Algorithmus. Dafür ist es notwendig den Beobachtungsraum sowie den Aktionsraum zu neu zu definieren.

Zur **Initialisierung** werden zum einen alle Parameter zur Instanziierung des DualDroneAviary Objekt erwartet. Zum Anderen sind zusätzlich die Strategie und dessen Rauschverhalten der angreifenden Drohne anzugeben, sodass diese in Klassenvariablen gespeichert werden. Auf der Ebene des SingleDroneWrapper-Klasse wird der **Beobachtungsraum** mit allen Informationen beider Quadrokopter zu einem 40-elementigen Vektor zusammengefasst. Der **Handlungsraum** hingegen umfasst auf dieser Ebene lediglich einen elementigen Vektor mit dem bekannten Wertebereich. Zur Ausführung eines Simulationsschrittes werden in der **Step-Funtkion** die Aktion der Angreiferstrategie und die übergebene Aktion in ein Datenobjekt überführt. Anschließend kann mittels diesen Datenobjektes die Schrittfunktion der Elternklasse aufgerufen, und deren Rückgabewerte an den zu optimierenden Algorithmus weitergereicht werden. Als **Belohnungswert** wird dabei lediglich auf die im Trainingsszenario relevante Kompontente des Belohnungsobjektes zugegriffen.

Fasst man die beschriebene Umsetzung der Simulationsumgebung zusammen, so kann folgendes Klassendiagramm und deren Funktionsabhängigkeiten in Abbildung Zehn betrachtet werden.

3.3.2 Programmumsetzung des Optimierungsverfahrens

Mit der Entwicklung der Simulationsumgebung stehen Trainingsdaten zur Verfügung um darauf aufbauend Strategien des verstärkenden Lernens zu optimieren. Der gesamte Optimierungsprozess unterteilt sich in den Einsatz von imitierenden Lernen zu Beginn und folgender Optimierung

[229] Eigene Darstellung

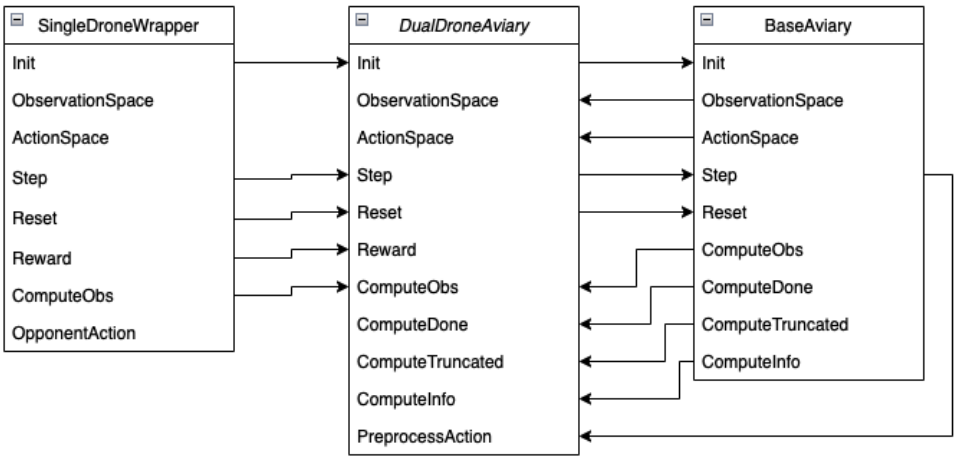


Abb. 10: Klassendiagramm mit Funktionabhängigkeiten[229]

mittels RL-Algorithmen. Die Umsetzung des Optimierungsverfahrens wird innerhalb der Python Dateien *imitation_train.py* und *train.py* beschrieben.

Das imitierende Training beginnt mit dem Herstellen des Trainingsszenarios zum Akquirieren der Demonstrationen des Experten. Innerhalb einer Trainingsiteration wird die Simulation unter den Trainingsgegebenheiten in **X** Episoden ausgeführt. Alle **aggr_physics_timestep** Zeitschritte wird die aktuelle Zustandsveränderung als Transition-Objekt der verwendeten Softwarebibliothek Imitation gespeichert. Dieses Datenobjekt beschreibt einen durch die Aktion entstehenden Übergang vom bekannten zu einem neuen beobachteten Zustand sowie den Erhalt der Belohnung. Hieraus ergeben sich eine Menge von aufeinanderfolgenden oder zeitversetzten Transition-Objekten. Die Übergänge in den Trainingsdaten liegen häufig nicht entsprechend ihrer Entstehung geordnet, um die Datenvielfalt zu erhöhen und korrelierende Effekte zu mindern. Basierend auf diesen Transaktionsdaten wird in **X** Epochen die Parameter der zu optimierenden Strategie angepasst. Die Anpassung der Gewichte einer Akteur-Kritiker Strategie wird durch den Behavioral Cloning Algorithmus vorgenommen. Insgesamt wird dieser abwechselnde Vorgang der Erhebung von Daten und die zur Basis dessen stattfindende Optimierung so häufig wiederholt, bis alle Iterationen die Menge an Expertendemonstrationen chargenweise abgearbeitet haben.

Als Experte wird eine regelbasierte Version der zu optimierenden Drohne eingesetzt. Alle regelbasierten Strategien sind in der *rule-based.py* Datei deklariert und umfassen die verteidigende Drohne und die Trainings- und Testversion der anzugreifenden Drohne. Die verteidigende Drohne verfolgt regelbasiert die direkte Flugbahn zu einem Vorhaltepunkt in Bewegungsrichtung der Angreiferdrohne. Der Vorhaltepunkt rückt mit geringer werdender Distanz zur generischen Drohne immer näher zu dessen aktueller Position. Die angreifende Drohne wird im ersten Trainingsszenario regelbasiert gesteuert, indem sie sich von ihrem Startpunkt in gerader Fluglinie zum Zielpunkt begibt. Dessen Erweiterung zur parabelförmigen Fluglinie wird in der nachfolgenden Sektion behandelt.

Wurde ein Akteur-Kritiker Modell durch imitierendes Lernen hin zum regelbasiertem Verhalten

optimiert, wird dieses als Basis zur weiteren Optimierung mittels RL verwendet. Mit dem Einsatz des *train.py* Programms lässt sich das in einer Pickle-Datei gespeicherte Modell laden. Dazu wird das grundlegende Neuronale Netz des PPO-Objekts mit der selben Struktur der Strategie aus dem imitierenden Lernen initialisiert. Anschließend werden die Gewichte und Vorurteile in ein Objekt des PPO-Algorithmus von Stable-Baselines3 übertragen. Mittels der learn-Funktion kann die erzeugte Strategie im gegebenen Trainingsszenario weiter optimiert werden. Durch die Tensorboard Bibliothek werden während des Trainingsprozesses gemittelte Metriken über alle Episoden berechnet sowie alle 10000 Zeitschritte eine Evaluation durchgeführt. In der Evaluationsperiode wird die aktuelle Strategie bemessen und dessen Episodenlänge, erzielte Belohnung und Erfolg aufgezeichnet. Als erfolgreich wird die Simulation gekennzeichnet, sobald die Strategie der Verteidigerdrohne die angreifende Drohne durch Kollision abfangen konnte. Anschließend an die Optimierung werden die neuen Modelle in Pickle Dateien gespeichert.

3.3.3 Programmumsetzung zur Laborexperiments

Als letzter Teil der Umsetzung wird die Durchführung des Experiments betrachtet. Kernaufgabe dieses Teils ist die Erhebung und Auswertung der Messdaten zu dem in der Einleitung des Kapitels verfassten Experimentaufbau. Die Erhebung der Testdaten erfolgt durch das *test.py*, die Auswertung durch das *experiment_ evaluation.py* Skript.

Mit der Ausführung des ersten Skripts wird ein Testklasse instanziiert und darin enthalten die Simulation nach den Eigenschaften des Testszenarios erzeugt. Das Testszenario implementiert dabei die Anforderung der Randomisierung des Zielpunktes. Zur zufälligen Bestimmung des Zielpunkts wird ebenso wie zur Initialisierung der Verteidigerdrohne, eine uniforme Verteilung bis zu 1,5 Meter um den Ursprung verwendet. Eine uniforme Verteilung fördert dabei die Varianz der Testszenarien und erhöht so die Validität des Experiments. Anders als bei der Initialisierung der Drohnen wird die Höhe des Zielpunktes stets auf 0,5 Meter festgelegt, um so Bodenkontakt und Simulationserfolg abzugrenzen. Zusätzlich zur Randomisierung des Zielpunktes wird im Testszenario der regelbasierte angreifende Quadrokopter im Aspekt seiner parabelförmigen Anflugsstrategie verändert. Ausgehend von den Startkoordinaten des Angreifers werden dazu insgesamt fünf Wegpunkte entlang einer unten geöffneten Parabel berechnet, welche nacheinander angeflogen werden. Die Parameter der Parabel berechnen sich wie in den Formeln Acht bis Zehn dargestellt.

$$a = -1/(2 * (\text{Distanz zum Zielpunkt}/6)) \quad (8)$$

$$b = 0 \quad (9)$$

$$c = \text{Z-Koordinate des Startpunktes} \quad (10)$$

Alle Messdaten werden aus der hundertfachen Durchführung der Testsimulation erhoben. Zu jedem Zeitschritt jeder Episode wird durch das RL-Modell eine bestimmte Aktion ausgeübt und

Episode	Episodenlänge	kumulierte Belohnung	Belohnungsabweichung zum Mittelwert	Episodenmisserfolg
1	1931	13512	3192	0
2	1579	8413	1873	1
3	1431	16920	6308	0
4	1944	6741	3701	1
5	1508	10173	89	0

Tab. 6: Beispieltabelle der erhobenen aggregierten Messdaten

eine Teilmenge dessen Resultats gemessen. Die Teilmenge umfasst die durch die Aktion unmittelbar ausgelöste Belohnung und einen Wahrheitswert, ob mit Aktion ein Misserfolg der Simulation eingegangen wurde. Beide Werte werden mit der aktuellen Episode und dem Zeitschritt einer Tabelle als Datensatz hinzugefügt. Nach dem erfolgreichem Durchlaufen aller Zeitschritte aller der 100 Episoden wird die Tabelle als Komma separierte Wertedatei (CSV) gespeichert.

Die erstellte Datei wird anknüpfend mittels *experiment_ evaluation.py* ausgewertet, indem die Messdaten geladen, die Signifikanztests durchgeführt und daran die Hypothesen bestimmt werden. Für jeden diesen drei Schritte wurde eine Funktion entwickelt, welche im Skript aufgerufen wird. Zum Laden der Messdaten sind die Speicherpfade zweier Wertedateien anzugeben, woraufhin der Verlauf der Kennzahlen zu einzelnen Episoden aggregiert werden. Dabei wird zunächst die Episodenlänge, die kumulierte Belohnung und die Anzahl an Misserfolgen ermittelt. Zu dieser Basis kann die Abweichung der Belohnung je Episode zu dem Mittel aller kumulierten Belohnungen im Testprozess berechnet werden. Nach dem Laden und Vorverarbeiten entspricht der Aufbau der Messdaten zweier Tabellen ähnlich der Tabelle Sechs.

Im nächsten Schritt ist mittels statistischer Signifikanztests die Ähnlichkeit zu einer Normalverteilung sowie die ungleiche Verteilung und mögliche Verbesserung aller Metriken zu untersuchen. Die Implementierung der Signifikanztests wird im Rahmen dieser Arbeit durch die Softwarebibliothek SciPy bereitgestellt. Iterativ werden für alle Metriken beider Messtabellen die Ähnlichkeit zur Normalverteilung überprüft, um zwischen einem T-Test und einem Mann-Whitney U Test auszuwählen. Anschließend wird je Metrik der entsprechende Test zur Ungleichheit und Verbesserung der Verteilung ausgeführt, woraus je Test eine Teststatistik und ein P-Wert gespeichert wird. Liegt der P-Wert eines Tests unter dem Signikanzniveau von 10% wird die H0 Hypothese abgelehnt und daraus schlussfolgernd die Ungleichheit oder Verbesserung der Metrik angenommen. Insgesamt wird diese Auswertung der Leistungsdaten zweier Strategien zweimal durchgeführt. Dabei werden die erzielten Messdaten zwischen der Strategien aus Trainingsszenario eins und drei sowie aus Szenario eins und vier betrachtet. Der vollstände Prozess der Experiments kann auch als BPMN Modell wie in Abbildung Elf dargestellt werden.

[229] Eigene Darstellung
[230] Eigene Darstellung

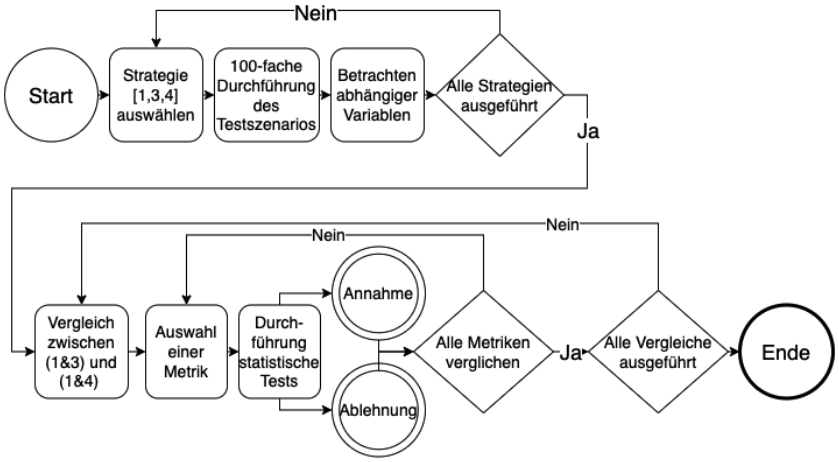


Abb. 11: BPMN Prozess des Laborexperiments[230]

4 Ergebnisse des Laborexperiments

5 Reflexion und Forschungsausblick

Anhang

Anhangverzeichnis

Anhang 1: Interview Transkripte

Anhang 1/1: Interview Transkript: Mitarbeiter eines Unternehmens

Literaturverzeichnis

ACM Digital Library (2/28/2023): ACM Digital Library. URL: https://dl.acm.org/.

Alghonaim, R./Johns, E. (2021): Benchmarking Domain Randomisation for Visual Sim-to-Real Transfer. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, S. 12802–12808. ISBN: 978-1-7281-9077-8. DOI: 10.1109/ICRA48506.2021.9561134.

Arulkumaran, K./Deisenroth, M. P./Brundage, M./Bharath, A. A. (2017): Deep Reinforcement Learning: A Brief Survey. In: *IEEE Signal Processing Magazine* 34.6, S. 26–38. DOI: 10.1109/MSP.2017.2743240.

Attia, A./Dayan, S. (2018): Global overview of Imitation Learning. arXiv: 1801.06503 [stat.ML].

Ayala, A./Cruz, F./Campos, D./Rubio, R./Fernandes, B./Dazeley, R. (2020): A Comparison of Humanoid Robot Simulators: A Quantitative Approach. In: *2020 Joint IEEE 10th International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*, S. 1–6. DOI: 10.1109/ICDL-EpiRob48136.2020.9278116.

Balakrishna, A./Thananjeyan, B./Lee, J./Li, F./Zahed, A./Gonzalez, J. E./Goldberg, K. (2020): On-Policy Robot Imitation Learning from a Converging Supervisor. In: *Proceedings of the Conference on Robot Learning*. Hrsg. von Leslie Pack Kaelbling/Danica Kragic/Komei Sugiura. Bd. 100. Proceedings of Machine Learning Research. PMLR, S. 24–41. URL: https://proceedings.mlr.press/v100/balakrishna20a.html.

Bellman, R. (1966): Dynamic Programming. In: *Science* 153.3731, S. 34–37. DOI: 10.1126/science.153.3731.34. eprint: https://www.science.org/doi/pdf/10.1126/science.153.3731.34. URL: https://www.science.org/doi/abs/10.1126/science.153.3731.34.

Bharadhwaj, H./Wang, Z./Bengio, Y./Paull, L. (2019): A Data-Efficient Framework for Training and Sim-to-Real Transfer of Navigation Policies. In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. DOI: 10.1109/icra.2019.8794310.

Brockman, G./Cheung, V./Pettersson, L./Schneider, J./Schulman, J./Tang, J./Zaremba, W. (2016): OpenAI Gym. In: *CoRR* abs/1606.01540. arXiv: 1606.01540. URL: http://arxiv.org/abs/1606.01540.

Canese, L./Cardarilli, G. C./Di Nunzio, L./Fazzolari, R./Giardino, D./Re, M./Spanò, S. (2021): Multi-Agent Reinforcement Learning: A Review of Challenges and Applications. In: *Applied Sciences* 11.11, S. 4948. DOI: 10.3390/app11114948. URL: https://www.mdpi.com/2076-3417/11/11/4948.

Chen, X./Hu, J./Jin, C./Li, L./Wang, L. (2021): Understanding Domain Randomization for Sim-to-real Transfer. In: *CoRR* abs/2110.03239. arXiv: 2110.03239. URL: https://arxiv.org/abs/2110.03239.

Collins, J./Ketter, W. (2022): Power TAC: Software architecture for a competitive simulation of sustainable smart energy markets. In: *SoftwareX* 20, S. 101217. ISSN: 2352-7110. DOI: https://doi.org/10.1016/j.softx.2022.101217. URL: https://www.sciencedirect.com/science/article/pii/S2352711022001352.

Cutler, M./Walsh, T. J./How, J. P. (2014): Reinforcement learning with multi-fidelity simulators. In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. DOI: 10.1109/icra.2014.6907423.

Deshpande, A. M./Kumar, R./Minai, A. A./Kumar, M. (2020): Developmental Reinforcement Learning of Control Policy of a Quadcopter UAV with Thrust Vectoring Rotors. URL: https://arxiv.org/pdf/2007.07793.

Deshpande, A. M./Minai, A. A./Kumar, M. (2021): Robust Deep Reinforcement Learning for Quadcopter Control. In: *IFAC-PapersOnLine* 54.20, S. 90–95. ISSN: 24058963. DOI: 10.1016/j.ifacol.2021.11.158.

Fang, B./Jia, S./Guo, D./Xu, M./Wen, S./Sun, F. (2019): Survey of imitation learning for robotic manipulation. In: *International Journal of Intelligent Robotics and Applications* 3, S. 362–369.

Foronda, C. L. (2021): What Is Virtual Simulation? In: *Clinical Simulation in Nursing* 52, S. 8. ISSN: 18761399. DOI: 10.1016/j.ecns.2020.12.004.

Furrer, F./Burri, M./Achtelik, M./Siegwart, R. (2016): RotorS—A Modular Gazebo MAV Simulator Framework. In: *Robot Operating System (ROS): The Complete Reference (Volume 1)*. Hrsg. von Anis Koubaa. Cham: Springer International Publishing, S. 595–625. ISBN: 978-3-319-26054-9. DOI: 10.1007/978-3-319-26054-9_23. URL: https://doi.org/10.1007/978-3-319-26054-9_23.

Gao, Y./Vedula, S. S./Reiley, C. E./Ahmidi, N./Varadarajan, B./Lin, H. C./Tao, L./Zappella, L./Béjar, B./Yuh, D. D. u. a. (2014): Jhu-isi gesture and skill assessment working set (jigsaws): A surgical activity dataset for human motion modeling. In: *MICCAI workshop: M2cai*. Bd. 3. 3.

GitHub (4/4/2023): Farama-Foundation/Gymnasium: A standard API for single-agent reinforcement learning environments, with popular reference environments and related utilities (formerly Gym). URL: https://github.com/Farama-Foundation/Gymnasium.

Google Scholar (2/28/2023). URL: https://scholar.google.de/.

Haarnoja, T./Zhou, A./Abbeel, P./Levine, S. (2018): Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In: *CoRR* abs/1801.01290. arXiv: 1801.01290. URL: http://arxiv.org/abs/1801.01290.

Hentati, A. I./Krichen, L./Fourati, M./Fourati, L. C. (2018): Simulation Tools, Environments and Frameworks for UAV Systems Performance Analysis. In: *2018 14th International Wireless Communications & Mobile Computing Conference (IWCMC)*. IEEE. DOI: 10.1109/iwcmc.2018.8450505.

Holzweißig, K. (2022): Wissenschaftliches Arbeiten. In: 6.6.

Hsu, K.-C./Ren, A. Z./Nguyen, D. P./Majumdar, A./Fisac, J. F. (2023): Sim-to-Lab-to-Real: Safe reinforcement learning with shielding and generalization guarantees. In: *Artificial Intelligence* 314, S. 103811. ISSN: 0004-3702. DOI: https://doi.org/10.1016/j.artint.2022.103811. URL: https://www.sciencedirect.com/science/article/pii/S0004370222001515.

Huang, S. H./Papernot, N./Goodfellow, I. J./Duan, Y./Abbeel, P. (2017): Adversarial Attacks on Neural Network Policies. In: *CoRR* abs/1702.02284. arXiv: 1702.02284. URL: http://arxiv.org/abs/1702.02284.

Hussein, A./Gaber, M. M./Elyan, E./Jayne, C. (2017): Imitation Learning: A Survey of Learning Methods. In: *ACM Comput. Surv.* 50.2. ISSN: 0360-0300. DOI: 10.1145/3054912. URL: https://doi.org/10.1145/3054912.

IEEE Xplore (2/28/2023). URL: https://ieeexplore.ieee.org/Xplore/home.jsp.

Ivaldi, S./Padois, V./Nori, F. (2014): Tools for dynamics simulation of robots: a survey based on user feedback. URL: https://arxiv.org/pdf/1402.7050.

Koch, W./Mancuso, R./West, R./Bestavros, A. (2018): Reinforcement Learning for UAV Attitude Control. In: *CoRR* abs/1804.04154. arXiv: 1804.04154. URL: http://arxiv.org/abs/1804.04154.

Körber, M./Lange, J./Rediske, S./Steinmann, S./Glück, R. (2021): Comparing Popular Simulation Environments in the Scope of Robotics and Reinforcement Learning. URL: https://arxiv.org/pdf/2103.04616.

Li, Y. (2019): Reinforcement Learning Applications. DOI: 10.48550/ARXIV.1908.06973. URL: https://arxiv.org/abs/1908.06973.

Liu, Z./Guo, Z./Cen, Z./Zhang, H./Tan, J./Li, B./Zhao, D. (2023): On the Robustness of Safe Reinforcement Learning under Observational Perturbations. arXiv: 2205.14691 [cs.LG].

Maria, A. (1997): Introduction to modeling and simulation. In: *Proceedings of the 29th conference on Winter simulation - WSC '97*. New York, New York, USA: ACM Press. DOI: 10.1145/268437.268440.

Mnih, V./Badia, A. P./Mirza, M./Graves, A./Lillicrap, T. P./Harley, T./Silver, D./Kavukcuoglu, K. (2016): Asynchronous Methods for Deep Reinforcement Learning. In: *CoRR* abs/1602.01783. arXiv: 1602.01783. URL: http://arxiv.org/abs/1602.01783.

Mnih, V./Kavukcuoglu, K./Silver, D./Graves, A./Antonoglou, I./Wierstra, D./Riedmiller, M. A. (2013): Playing Atari with Deep Reinforcement Learning. In: *CoRR* abs/1312.5602. arXiv: 1312.5602. URL: http://arxiv.org/abs/1312.5602.

Molchanov, A./Chen, T./Hönig, W./Preiss, J. A./Ayanian, N./Sukhatme, G. S. (2019): Sim-to-(Multi)-Real: Transfer of Low-Level Robust Control Policies to Multiple Quadrotors. In: *CoRR* abs/1903.04628. arXiv: 1903.04628. URL: http://arxiv.org/abs/1903.04628.

Moos, J./Hansel, K./Abdulsamad, H./Stark, S./Clever, D./Peters, J. (2022): Robust Reinforcement Learning: A Review of Foundations and Recent Advances. In: *Machine Learning and Knowledge Extraction* 4.1, S. 276–315. ISSN: 2504-4990. DOI: 10.3390/make4010013. URL: https://www.mdpi.com/2504-4990/4/1/13.

Ningombam, D. D. (2022): Deep Reinforcement Learning Algorithms for Machine-to-Machine Communications: A Review. In: *2022 13th International Conference on Computing Communication and Networking Technologies (ICCCNT)*. IEEE. DOI: 10.1109/icccnt54827.2022.9984457.

Pan, A./Lee, Y./Zhang, H./Chen, Y./Shi, Y. (2021): Improving Robustness of Reinforcement Learning for Power System Control with Adversarial Training. In: *arXiv e-prints*, arXiv:2110.08956, arXiv:2110.08956. DOI: 10.48550/arXiv.2110.08956. arXiv: 2110.08956 [eess.SY].

Panerati, J./Zheng, H./Zhou, S./Xu, J./Prorok, A./Schoellig, A. P. (2021): Learning to Fly – a Gym Environment with PyBullet Physics for Reinforcement Learning of Multi-agent Quadcopter Control. URL: https://arxiv.org/pdf/2103.02142.

Pinto, L./Davidson, J./Sukthankar, R./Gupta, A. (2017): Robust Adversarial Reinforcement Learning. In: *CoRR* abs/1703.02702. arXiv: 1703.02702. URL: http://arxiv.org/abs/1703.02702.

Pullum, L. L. (2022): Review of Metrics to Measure the Stability, Robustness and Resilience of Reinforcement Learning. arXiv: 2203.12048 [cs.LG].

Raffin, A./Hill, A./Gleave, A./Kanervisto, A./Ernestus, M./Dormann, N. (2021): Stable-Baselines3: Reliable Reinforcement Learning Implementations. In: *J. Mach. Learn. Res.* 22.1. ISSN: 1532-4435.

Recker, J. (2021): Scientific research in information systems: A beginner's guide. Second Edition. Progress in IS. Cham: Springer International Publishing. ISBN: 9783030854362. URL: https://ebookcentral.proquest.com/lib/kxp/detail.action?docID=6789173.

Reda, D./Tao, T./van de Panne, M. (2020): Learning to Locomote: Understanding How Environment Design Matters for Deep Reinforcement Learning. In: *Motion, Interaction and Games*. Hrsg. von Daniele Reda/Tianxin Tao/Michiel van de Panne. New York, NY, USA: ACM, S. 1–10. DOI: 10.1145/3424636.3426907.

Sadeghi, F./Levine, S. (2016): CAD2RL: Real Single-Image Flight without a Single Real Image. In: *CoRR* abs/1611.04201. arXiv: 1611.04201. URL: http://arxiv.org/abs/1611.04201.

Schott, L./Hajri, H./Lamprier, S. (2022): Improving Robustness of Deep Reinforcement Learning Agents: Environment Attack based on the Critic Network. In: *2022 International Joint Conference on Neural Networks (IJCNN)*, S. 1–8. DOI: 10.1109/IJCNN55064.2022.9892901.

Schuderer, A./Bromuri, S./van Eekelen, M. (2021): Sim-Env: Decoupling OpenAI Gym Environments from Simulation Models. In: *International Conference on Practical Applications of Agents and Multi-Agent Systems*. Springer, Cham, S. 390–393. DOI: 10.1007/978-3-030-85739-4{\textunderscore}39. URL: https://link.springer.com/chapter/10.1007/978-3-030-85739-4_39.

Schulman, J./Levine, S./Moritz, P./Jordan, M. I./Abbeel, P. (2015): Trust Region Policy Optimization. In: *CoRR* abs/1502.05477. arXiv: 1502.05477. URL: http://arxiv.org/abs/1502.05477.

Schulman, J./Wolski, F./Dhariwal, P./Radford, A./Klimov, O. (2017): Proximal Policy Optimization Algorithms. In: *CoRR* abs/1707.06347. arXiv: 1707.06347. URL: http://arxiv.org/abs/1707.06347.

Shah, S./Dey, D./Lovett, C./Kapoor, A. (2017): AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles. In: *CoRR* abs/1705.05065. arXiv: 1705.05065. URL: http://arxiv.org/abs/1705.05065.

Silano, G./Iannelli, L. (2019): CrazyS: A Software-in-the-Loop Simulation Platform for the Crazyflie 2.0 Nano-Quadcopter. In: *Robot Operating System (ROS): The Complete Reference (Volume 4)*. Hrsg. von Anis Koubaa. Cham: Springer International Publishing, S. 81–115. ISBN: 978-3-030-20190-6. DOI: 10.1007/978-3-030-20190-6_4. URL: https://doi.org/10.1007/978-3-030-20190-6_4.

Slaoui, R. B./Clements, W. R./Foerster, J. N./Toth, S. (2019): Robust Domain Randomization for Reinforcement Learning. In: *CoRR* abs/1910.10537. arXiv: 1910.10537. URL: http://arxiv.org/abs/1910.10537.

Sutton, R. S./Barto, A. G. (2018): Reinforcement Learning, second edition: An Introduction. MIT Press. ISBN: 9780262352703.

Tobin, J./Fong, R./Ray, A./Schneider, J./Zaremba, W./Abbeel, P. (2017): Domain randomization for transferring deep neural networks from simulation to the real world. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, S. 23–30. DOI: 10.1109/IROS.2017.8202133.

Todorov, E./Erez, T./Tassa, Y. (2012): MuJoCo: A physics engine for model-based control. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, S. 5026–5033. DOI: 10.1109/IROS.2012.6386109.

Wang, Z./Hong, T. (2020): Reinforcement learning for building controls: The opportunities and challenges. In: *Applied Energy* 269, S. 115036. ISSN: 0306-2619. DOI: 10.1016/j.apenergy.2020.115036.

Webster, J./Watson, R. T. (2002): Analyzing the Past to Prepare for the Future: Writing a Literature Review. In: *MIS Q.* 26.2, S. xiii–xxiii. ISSN: 0276-7783.

Wong, A./Bäck, T./Kononova, A. V./Plaat, A. (2022): Deep multiagent reinforcement learning: challenges and directions. In: *Artificial Intelligence Review*. ISSN: 0269-2821. DOI: 10.1007/s10462-022-10299-x.

Yan Duan/Xi Chen/Rein Houthooft/John Schulman/Pieter Abbeel (2016): Benchmarking Deep Reinforcement Learning for Continuous Control. In: *International Conference on Machine Learning*, S. 1329–1338. ISSN: 1938-7228. URL: https://proceedings.mlr.press/v48/duan16.html.

Zhai, P./Hou, T./Ji, X./Dong, Z./Zhang, L. (2022): Robust Adaptive Ensemble Adversary Reinforcement Learning. In: *IEEE Robotics and Automation Letters* 7.4, S. 12562–12568. DOI: 10.1109/LRA.2022.3220531.

Zhang, A./Wu, Y./Pineau, J. (2018): Natural Environment Benchmarks for Reinforcement Learning. DOI: 10.48550/ARXIV.1811.06032. URL: https://arxiv.org/abs/1811.06032.

Zhao, W./Queralta, J. P./Westerlund, T. (2020): Sim-to-Real Transfer in Deep Reinforcement Learning for Robotics: a Survey. In: *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE. DOI: 10.1109/ssci47803.2020.9308468.

Erklärung

Ich versichere hiermit, dass ich meine Bachelorarbeit mit dem Thema: *Experiment zur Verbesserung der Robustheit von Reinforcement Learning Modellen in Drohnensimulationen anhand trainiertem Gegenspieler* selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

(Ort, Datum) (Unterschrift)