Weiterentwicklung, Test und Charakterisierung eines Reinforcement Learning basierten Reglers für ein Stickstoff-Experimentaltriebwerk

Development, Testing and Characterization of a Reinforcement Learning Based Controller for an Experimental Nitrogen Cold Gas Engine

Masterarbeit von
cand. aer. Christian Kley

IRS-Nr.: 22-S-013

Betreuer:
Prof. Dr.-Ing. Stefan Schlechtriem
M. Sc. Till Hörger

Deutsches Zentrum für Luft- und Raumfahrt,
Institut für Raumfahrtantriebe

Institut für Raumfahrtsysteme,
Universität Stuttgart

Juli 2022

UNIVERSITÄT STUTTGART
INSTITUT FÜR RAUMFAHRTSYSTEME
im Raumfahrtzentrum Baden-Württemberg (RZBW)

Pfaffenwaldring 29 · 70569 Stuttgart · Tel.: (0711) 685-62375 · Fax: (0711) 685-63596 · www.irs.uni-stuttgart.de

Aufgabenstellung Masterarbeit

für Herrn Christian Kley

Weiterentwicklung, Test und Charakterisierung eines Reinforcement Learning basierten Reglers für ein Stickstoff - Experimentaltriebwerk

Development, Testing and Characterization of a Reinforcement Learning Based Controller for an Experimental Nitrogen Cold Gas Engine

Motivation:

Heutzutage werden die meisten Flüssigraketentriebwerke lediglich mithilfe von vordefinierten Ventilsequenzen gesteuert oder verwenden in Ausnahmefällen im stationären Betriebspunkt eine zustandsabhängige Regelung. Durch die reine Steuerung des Hochfahrens sowie des Abschaltens des Triebwerks gibt es keine Möglichkeiten beim Auftreten von unvorhergesehenem Verhalten von Komponenten einzugreifen. Unvorhergesehenes Verhalten kann jedoch zu Beschädigungen von einzelnen Triebwerkskomponenten führen, welche besonders im Kontext der Langlebigkeit von Orbitalantrieben kritisch sind. Orbitalantriebe müssen für Einsatzzeiträume von mehr als 10 Jahren eine präzise Lage- und Bahnregelung des Raumfahrzeugs ermöglichen, daher sind Abweichungen von den nominellen Betriebsbedingungen unbedingt zu vermeiden. Moderne Regelalgorithmen haben das Potenzial eine zustandsabhängige Regelung aller Betriebsphasen zu ermöglichen und erhöhen weiterhin die Flexibilität in der Missionsplanung. Aus diesem Grund werden am DLR unterschiedliche, moderne Regelverfahren untersucht, u.a. auch Regler, welche Methoden des Maschinellen Lernens verwenden. Ein Beispiel hierfür ist das sogenannte Reinforcement Learning (RL).

Beim Reinforcement Learning findet das Training des neuronalen Netzes aufgrund der großen benötigten Datenmenge in einem geeigneten Simulator, zum Beispiel EcosimPro ESPSS, statt. Anschließend wird der Regler an die reale Anwendung übertragen. Dabei treten Unterschiede zwischen dem Regelverhalten in der Simulation und dem realen Prüfstand hervor. Im Rahmen der Arbeit sollen auf verschiedene Weise trainierte Regler am Prüfstand hinsichtlich ihrer Regelgüte und Robustheit qualitativ und quantitativ untersucht werden und ein Vergleich mit den Ergebnissen der Simulation gezogen werden.

Aufgabenstellung:

- Literaturrecherche zu bisherigen Arbeiten für RL-basierte Raketentriebwerksregler
- Literaturrecherche zur Charakterisierung verschiedener Regler/Bestimmung einer Regelgüte (Norm)
- Training verschiedener RL-basierter Regler für den bestehenden Versuchsaufbau mit einem Stickstoff Kaltgastriebwerk
- Untersuchung der Regelgüte und der Robustheit der RL-Regler am Prüfstand und Vergleich mit den in der Simulation erzielten Ergebnissen
- Verfassen der Arbeit, Dokumentation der Ergebnisse

Betreuer intern: Till Hörger, M.Sc.

Bearbeitungsbeginn: 02.11.2021

Einzureichen spätestens: 02.05.2022

Prof. Dr. S. Schlechtriem
(Verantwortlicher Hochschullehrer)

Empfangsbestätigung:
Ich bestätige hiermit, dass ich die Aufgabenstellung sowie die rechtlichen Bestimmungen und die Studien- und Prüfungsordnung gelesen und verstanden habe.

Unterschrift des/der Studierenden

Erklärungen

Hiermit versichere ich, **Kley, Christian,** dass ich diese ***Masterarbeit*** selbstständig mit Unterstützung des Betreuers / der Betreuer angefertigt und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe. Die Arbeit oder wesentliche Bestandteile davon sind weder an dieser noch an einer anderen Bildungseinrichtung bereits zur Erlangung eines Abschlusses eingereicht worden.

Ich erkläre weiterhin, bei der Erstellung der Arbeit die einschlägigen Bestimmungen zum Urheberschutz fremder Beiträge entsprechend den Regeln guter wissenschaftlicher Praxis[1] eingehalten zu haben. Soweit meine Arbeit fremde Beiträge (z.B. Bilder, Zeichnungen, Textpassagen etc.) enthält, habe ich diese Beiträge als solche gekennzeichnet (Zitat, Quellenangabe) und eventuell erforderlich gewordene Zustimmungen der Urheber zur Nutzung dieser Beiträge in meiner Arbeit eingeholt. Mir ist bekannt, dass ich im Falle einer schuldhaften Verletzung dieser Pflichten die daraus entstehenden Konsequenzen zu tragen habe.

Friedrichshafen, 07.06.2022, ...

Ort, Datum, Unterschrift

Hiermit erkläre ich mich damit einverstanden, dass meine ***Masterarbeit*** zum Thema:

Entwicklung und Test eines Reinforcement Learning basierten Reglers für ein Stickstoff Experimentaltriebwerk in der Institutsbibliothek des Instituts für Raumfahrtsysteme ohne Sperrfrist öffentlich zugänglich aufbewahrt und die Arbeit auf der Institutswebseite sowie im Online-Katalog der Universitätsbibliothek erfasst wird. Letzteres bedeutet eine dauerhafte, weltweite Sichtbarkeit der bibliographischen Daten der Arbeit (Titel, Autor, Erscheinungsjahr, etc.).

Nach Abschluss der Arbeit werde ich zu diesem Zweck meinem Betreuer neben dem Prüfexemplar eine weitere gedruckte sowie eine digitale Fassung übergeben.

Der Universität Stuttgart übertrage ich das Eigentum an diesen zusätzlichen Fassungen und räume dem Institut für Raumfahrtsysteme an dieser Arbeit und an den im Rahmen dieser Arbeit von mir erzeugten Arbeitsergebnissen ein kostenloses, zeitlich und örtlich unbeschränktes, einfaches Nutzungsrecht für Zwecke der Forschung und der Lehre ein. Falls in Zusammenhang mit der Arbeit Nutzungsrechtsvereinbarungen des Instituts mit Dritten bestehen, gelten diese Vereinbarungen auch für die im Rahmen dieser Arbeit entstandenen Arbeitsergebnisse.

Friedrichshafen, 07.06.2022, ...

Ort, Datum, Unterschrift

[1] Nachzulesen in den DFG-Empfehlungen zur „Sicherung guter wissenschaftlicher Praxis" bzw. in der Satzung der Universität Stuttgart zur „Sicherung der Integrität wissenschaftlicher Praxis und zum Umgang mit Fehlverhalten in der Wissenschaft"

Danksagung

Diese Masterarbeit entstand unter den besonderen Umständen der wieder aufkeimenden Corona-Pandemie 2022. Der Zugang zum DLR Standort Lampoldshausen wurde einen Monat nach Beginn meiner Tätigkeit eingeschränkt und viele Arbeiten mussten somit im Home-Office erledigt werden. Der fachliche Austausch im Rahmen der Arbeit fand meist nur via Online-Konferenz statt.

Ganz besonders bin ich daher für die intensive und dynamische Zusammenarbeit mit meinem Betreuer Till Hörger dankbar, der jederzeit für eine Besprechung in Skype zur Verfügung stand.

Mein Dank gilt weiterhin Herrn Professor Dr.-Ing. Stefan Schlechtriem für die Ermöglichung der Durchführung dieser Masterarbeit am Deutschen Zentrum für Luft- und Raumfahrt in Lampoldshausen.

Zuletzt möchte ich mich herzlich bei meiner Freundin Janine Dellwo und meiner ganzen Familie bedanken. Ihr wart mir während der Abschlussarbeit und während des gesamten Studiums eine große Stütze!

Christian Kley

Friedrichshafen, Juni 2022

Kurzfassung

Raketentriebwerke und Orbitalantriebe verwenden heutzutage meist voreingestellte Regelventilsequenzen zur Antriebssteuerung. Eine zustandsabhängige Regelung kommt in Ausnahmefällen nur im stationären Betriebspunkt zum Einsatz. Auf diese Weise kann auf unvorhergesehenes Verhalten von Komponenten im Betriebspunkt und während des Startvorgangs nicht reagiert werden. Eine mögliche Folge ist die Beschädigung von Triebwerkskomponenten, was insbesondere im Kontext der zu gewährleistenden Langlebigkeit von Orbitalantrieben kritisch ist. Um dieser Anforderung in Zukunft besser gerecht zu werden, sollen moderne Regelalgorithmen zur Triebwerksregelung zum Einsatz kommen. Solche Verfahren besitzen das Potenzial, eine durchgängig zustandsabhängige Regelung zu gewährleisten und die Flexibilität in der Missionsplanung zu erhöhen. Aus diesem Grund untersucht und erforscht das Deutsche Zentrum für Luft- und Raumfahrt e.V. (DLR) Modelle und Methoden des maschinellen Lernens. Ein vielversprechender Ansatz zur zustandsabhängigen Regelung zukünftiger Raumfahrtantriebe ist das *Reinforcement Learning* (RL).

RL basierte Regler benötigen große Datenmengen für das Training der verwendeten neuronalen Netze. Hierfür werden geeigneten Simulatoren eingesetzt. Im Rahmen dieser Arbeit ist der Aufbau eines Simulationsmodells eines Stickstoff-Experimentaltriebwerks in der Simulationsumgebung EcosimPro/ESPSS beschrieben. Dieses Modell wird mit einem in Python programmierten Reinforcement Learning Regler geregelt. Der Regler erlaubt es, alle physikalisch erreichbaren Betriebspunkte des Triebwerks anzusteuern. Anschließend wird der Regler in die reale Anwendung am Prüfstand übertragen. Das Simulationsmodell wurde hierfür zunächst anhand von Messdaten des realen Prüfstandes validiert.

Um Modellierungsfehler zwischen Simulation und Realität zu minimieren, werden auf verschiedene Weise trainierte Regler hinsichtlich ihrer Regelgüte und Robustheit qualitativ und quantitativ untersucht. Zu diesem Zweck werden 42 ausgewählte neuronale Netze trainiert und getestet, um unterschiedliche Belohnungsfunktionen und Beobachtungsräume zu untersuchen. Es werden verschiedene Techniken wie beispielsweise die *Domain Randomization* angewendet, um die Robustheit der Regler zu erhöhen. Zusätzlich werden zufällige Störungen und Variationen verschiedener Prozessgrößen in das Training eingebracht und eine umfassende Studie von Testparametern durchgeführt.

Das Ergebnis der Arbeit ist die Regelung eines bisher ungeregelten Prüfstands für Orbitalantriebe mittels Reinforcement Learning. Zusätzlich dient der erlangte Erkenntnisgewinn beim Training der verwendeten neuronalen Netze zukünftigen Forschungsaufgaben in diesem Bereich.

Abstract

Nowadays, rocket engines and orbital propulsion systems mostly use preset control valve sequences for propulsion control. Closed loop control is used in exceptional cases only at the steady-state operating point. In this way, it is not possible to react to unforeseen behavior of components at the operating point and during the startup process. One possible consequence is damage to engine components, which is particularly critical in the context of the longevity of orbital propulsion systems. To better meet this requirement in the future, modern control algorithms are to be used. These have the potential to ensure closed loop control throughout the whole engine operating range and can increase flexibility in mission planning. For this reason, the German Aerospace Center (DLR) is investigating and researching machine learning models and methods. One promising approach for closed loop control of future space propulsion systems is reinforcement learning (RL).

RL based controllers require large amounts of data for training the neural networks used. Suitable simulators are used for this purpose. Within the scope of this work, the construction of a simulation model of a nitrogen experimental engine in the simulation environment EcosimPro/ESPSS is described. This model is controlled with a reinforcement learning controller programmed in Python. The controller allows to control all physically achievable operating points of the engine. The controller is then transferred to the real application on the test rig. The simulation model was first validated using measurement data from the real test bench.

In order to minimize modeling errors between simulation and reality, controllers trained in different ways are examined qualitatively and quantitatively with respect to their control performance and robustness. For this purpose, 42 selected neural networks are trained and tested to investigate different reward functions and observation spaces. Different techniques such as „domain randomization“ are applied to increase the robustness of the controllers. In addition, random perturbations and variations of process variables are introduced into the training and a comprehensive study of test parameters is conducted.

The result of the work is the control of a previously uncontrolled test bench for orbital drives using reinforcement learning. In addition, the knowledge gained in training neural networks will serve future research tasks in this area.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Nomenklatur

Lateinische Zeichen

A	Raum aller möglichen Aktionen	
a	Aktion	
$\mathcal{B}$	Zufällig ausgewählte Transitionen aus dem Replay Buffer	
$\mathcal{D}$	Replay Buffer	
d_h	Hydraulischer Durchmesser	[m]
$\mathbb{E}$	Erwartungswert	
e	Vektor, der Informationen eines Zeitschritts enthält	
$\mathcal{F}$	Set von Trajektorien	
F	Schub	[N]
G	Return, summierter Reward	
H	Entropie	
J	Erwarteter Return einer Trajektorie	
k_v	Durchflussbeiwert	[m^3/h]
L	Fehlergleichung	
$\dot{m}$	Massenstrom	[kg/s]
N	Anzahl	
P	Druck	[Pa]
$\mathcal{P}$	Zustandsübergangsfunktion	
Q	Action-Value-Funktion	
R	Raum aller möglichen Rewards	
r	Reward	
s	Zustand	
S	Raum aller möglichen Zustände	
T	Zeit	[s]
V	Value-Funktion	
y	Zielfunktion	

Griechische Zeichen

α	Entropie-Regulierungskoeffizient
β	Lernrate
γ	Discount-Faktor
ϵ	Exploration-Noise, Zufallszahl [0,1]
ζ	Widerstandsziffer
Θ	Gewichte des neuronalen Netzes zur Policy-Approximation
μ	Vektor mit zu variierenden Größen
π	Policy, Entscheidungsregel
τ	Trajektorie
Φ	Gewichte des neuronalen Netzes zur Q-Approximation
∇	Nabla-Operator

Indizes

alt	Ausgangswert einer Iteration
BK	Brennkammer
DR	Druckregler
ist	Istwert einer Größe
neu	Neue Approximation eines Werts
RV	Regelventil
$soll$	Sollwert einer Größe
t	Zeitpunkt
$t+1$	Darauffolgender Zeitpunkt
$*$	Optimaler Wert
$'$	Neuer Wert

Abkürzungen

API	Application Programming Interface
CuCrZr	Kupfer-Chrom-Zirkon
DA	Domain Adaption
DDPG	Deep Deterministic Policy Gradient
DLR	Deutsches Zentrum für Luft- und Raumfahrt e.V.
DQN	Deep Q-Network
DR	Domain Randomization
DRL	Deep Reinforcement Learning
ESA	European Space Agency
ESPSS	European Space Propulsion System Simulation
GUI	Graphical User Interface

HyNOx	Hydrocarbons and Nitrous Oxide
KI	Künstliche Intelligenz
MDP	Markov Decision Process
MMH	Monomethylhydrazin
NASA	National Aeronautics and Space Administration
PID	Proportional-Integral-Derivative
PPO	Proximal Policy Optimization
RL	Reinforcement Learning
SAC	Soft Actor Critic
TD	Time Difference
TD3	Twin Delayed Deep Deterministic Policy Gradient
UDMH	Unsymmetrisches Dimethylhydrazin

1. Einleitung

1.1. Hintergrund der Thematik

In den vergangenen Jahren sind die Anforderungen an die Steuerung von Raumflugkörpern, insbesondere im Bezug auf wiederverwendbare Triebwerke, erheblich gestiegen [1]. Der erste große Durchbruch in dieser Entwicklung erzielte SpaceX mit dem Start und der anschließenden Landung der ersten Stufe ihrer Falcon 9 Rakete im Jahr 2015. Insbesondere die Möglichkeit der Wiederverwendung von Raketenkomponenten verspricht Kosteneinsparungen, verschafft einen Wettbewerbsvorteil und eröffnet darüber hinaus neue Möglichkeiten im Bereich der kommerziellen Weltraumfahrt. In Europa wird seither mit CALLISTO ebenfalls an einem Demonstrator für wiederverwendbare Trägerraketen gearbeitet [2].

Komplexe Missionsszenarien wie beispielsweise Manöver in der Umlaufbahn oder die Fähigkeit, senkrecht zu starten und zu landen, erfordern eine präzise Triebwerksregelung zur Schubkontrolle und die Wiederzündfähigkeit von Triebwerken [3]. Aufgrund von Rußablagerungen, erhöhter Leckage-Massenströme und durch Dichtungsalterung oder Erosion der Turbinenschaufeln kann die Leistung der Triebwerkskomponenten mit der Zeit abnehmen. Der kosteneffiziente Betrieb einer wiederverwendbaren Trägerrakete ist daher nur möglich, wenn die Triebwerke eine lange Lebensdauer ohne teure Wartung besitzen [4].

Das von der Ariane 5 Rakete der europäischen Raumfahrtagentur ESA verwendete Vulcain Triebwerk wird wie die meisten Triebwerke ihrer Generation nicht geregelt, sondern lediglich mit vorgegebenen Ventilsequenzen gesteuert. Auf diese Weise kann zwar ein optimaler Betriebspunkt im Vorfeld eingestellt, allerdings nicht auf unvorhergesehene Ereignisse während einer Mission reagiert werden. Mit einer voreingestellten Ventilsequenz kann es daher beim Wiederverwenden der Antriebssysteme zu Abweichungen vom geplanten Betriebspunktes des Triebwerks kommen, da nicht auf die oben beschriebenen Systemänderungen reagiert wird. Dies kann zu einem nicht optimalen Betrieb des Triebwerks und schlimmstenfalls zum Absturz der Rakete in kritischen Missionsphasen führen [5].

In den letzten Jahren hat sich ein enormes Interesse an der Anwendung von Methoden der künstlichen Intelligenz, insbesondere von Algorithmen des maschinellen Lernens, gezeigt. Dies hat große Auswirkungen auf die Luft- und Raumfahrttechnik im Allgemeinen [6]. In Simulationen wurde am DLR bereits gezeigt, dass Reinforcement Learning basierte Regler den Startvorgang und den Betrieb eines Vulcain 1 Triebwerks besser als ein PID-Regler regeln und somit auf sich ändernde Systemdynamiken reagieren können [3]. Der zustandsabhängige Regler ist in diesem Fall ein künstliches neuronales Netz,

welches mit Deep Reinforcement Learning (DRL) trainiert wurde. Ein solcher Regler soll im weiteren Verlauf der Forschung an Raketenprüfständen eingesetzt werden, um zu demonstrieren, dass eine optimale, zustandsabhängige Regelung mit RL möglich ist.

Bisherige Anwendungen von RL basierter künstlicher Intelligenz lassen sich schon in den vergangenen Jahrzehnten hauptsächlich in der Computerspielbranche finden [7]. Bei Videospielen ist es oft üblich, dass Spieler gegen eine Form der künstlichen Intelligenz antreten. Der Firma OpenAI gelang es im Rahmen ihres Projektes *OpenAI Five* in den Jahren 2016-2019 die damaligen Weltmeister in dem weit verbreiteten und höchst komplexen Computerspiel Dota 2 mithilfe ihrer für diesen Zweck trainierten künstlichen Intelligenz (KI) zu bezwingen. Das Erfolgsrezept bestand aus einem Reinforcement-Learning-Agenten, der innerhalb weniger Monate mit einer entsprechenden Spielzeit von 45.000 Jahren trainiert wurde - ein für ein menschliches eSport-Team unmögliches Trainingspensum [8]. Ein weiteres prominentes Beispiel einer industrienahen Anwendung ist eine Roboterhand, die in der Lage ist, einen Zauberwürfel zu lösen [9]. Allerdings stehen RL basierte Anwendungen in der realen Welt noch vor einigen Herausforderungen, welche sich in folgende drei Bereiche einteilen lassen können [5, 10].

Probeneffizienz RL-basierte neuronale Netze benötigen sehr viele Beispiele um ein bestimmtes Verhalten zu lernen. Während es in Simulationen möglich ist, durch Parallelisierung viele Trainingsdaten zu erzeugen, ist dies in der Realität nur unter hohem Zeit- und Kosteneinsatz möglich.

Betriebsbereichseinschränkungen In der Realität dürfen sich die Betriebsgrößen eines Systems nur in bestimmten Grenzen bewegen, andernfalls kann dies fatale Folgen haben. In Simulationen ist sind Betriebsbereichsüberschreitung unproblematisch, da der Vorgang einfach neu gestartet werden kann. Dies ist relevant, da die Überschreitung ungewollter Betriebsbereiche für den Lernprozess notwendig ist.

Robustheit Der Begriff Robustheit bezeichnet die Fähigkeit, auf Störungen im System entsprechend reagieren und die Zielsetzung weiterhin erfüllen zu können.

Diese Herausforderungen sind Teil der „Sim-to-Real-Gap“-Problematik, einer Lücke bei der Übertragung von der Simulation in die reale Anwendung [11].

1.2. Zielsetzung und Aufbau der Arbeit

Es ist das Ziel dieser Arbeit, die im vorigen Abschnitt beschriebene Lücke zwischen Simulation und Realität für den Anwendungsfall einer Raketentriebwerksregelung zu verkleinern und Erkenntnisse aus dem Training neuronaler Netze für zukünftige Forschungsaufgaben zu gewinnen. Hierfür wird ein mit Deep Reinforcement Learning (DRL) erzeugter Regelalgorithmus für einen experimentellen Orbitalantriebes in der Simulation und am Prüfstand bewertet. Die Versuchsbrennkammer soll in einem Versuchsaufbau mit Stickstoff betrieben werden. Die Versuche finden am Prüfstandskomplex M11 des Deutschen Zentrums für Luft- und Raumfahrt in Lampoldshausen statt [12].

Im folgenden Kapitel werden Orbitalantriebe und die für das Reinforcement Learning fundamentalen Gleichungen und Prinzipien vorgestellt und weitere für die Arbeit relevante Konzepte erörtert. Dazu zählt unter anderem die Methode der *Domain Randomization*. Kapitel 3 beschäftigt sich zunächst mit dem Prüfstandsaufbau und der Modellbildung des untersuchten Experimentaltriebwerks in der Simulationssoftware EcoSimPro/ESPSS. Anschließend wird auf die in der Arbeit implementierte Programmstruktur in Python zur Steuerung des Trainings der RL-Regler eingegangen. Die Simulationsergebnisse und deren Vergleich mit den Testergebnissen am Prüfstand werden in Kapitel 4 präsentiert. Abschließend wird im letzten Kapitel ein Fazit und die daraus resultierenden Schlussfolgerungen für weitere Untersuchungen gezogen.

2. Theoretische Grundlagen

Nach einer kurzen Einführung in die Orbitalantriebe und deren Abgrenzung zu anderen Antriebssystemen folgt in diesem Kapitel eine Darstellung der theoretischen Grundlagen zum Verständnis der angewendeten Methoden. Neben einer anschaulichen Erklärung über das Reinforcement Learning mit seinen Vorteilen und Herausforderungen werden die relevanten mathematischen Modelle und Gleichungen näher beschrieben. Abschließend wird der in dieser Arbeit verwendete Algorithmus vorgestellt.

2.1. Orbitale Antriebssysteme

Unter dem Begriff der Orbitalantriebe werden sekundäre Antriebssysteme von Raumflugkörpern verstanden, deren Aufgabe es ist, deren Lage und Bahnregelung zu übernehmen. Im Gegensatz zu den Hauptantriebssystemen besitzen Orbitalantriebe einen geringeren Schub und ein geringeres Antriebsvermögen. Um ihre missionsbedingten Anforderungen zu erfüllen, kennzeichnen sich sekundäre Antriebssysteme unter anderem durch ihre Langlebigkeit, Wiederzündbarkeit und Kaltstartfähigkeit aus [13]. Als gebräuchliches Treibstoffsystem haben sich in den 1960er Jahren Hydrazin und dessen Derivate Unsymmetrisches Dimethylhydrazin (UDMH) und Monomethylhydrazin (MMH) als geeignet erwiesen [14]. Aufgrund der hochgradigen Gesundheitsschädlichkeit von Hydrazin steht seit 2011 ein Verbot in der Europäischen Union zur Diskussion [15]. Aus diesem Grund erforscht das DLR in Lampoldshausen Alternativen, die den hohen Anforderungen an Lagerfähigkeit, Antriebsvermögen und Zuverlässigkeit gerecht werden. Ein aktuelles Beispiel hierfür ist die Kombination aus Lachgas und Kohlenwasserstoffen, kurz HyNOx (*Hydrocarbons and Nitrous Oxide*). Derartige Treibstoffe werden als grüne Treibstoffe (*Green Propellants*) bezeichnet [16].

Es konnte bereits gezeigt werden, dass eine regenerativ gekühlte Versuchsbrennkammer mit einer Lachgas/Ethan Treibstoffmischung im Mono- und Bipropellantbetrieb im stationären Zustand betrieben werden kann [17]. Das Lachgas wird in Gasflaschen im unter Druck verflüssigten Gaszustand gelagert. Durch den vergleichsweise hohen Dampfdruck von 50,6 bar bei 20 °C ist es möglich, das Gas direkt von einem Tank in die Brennkammer des Triebwerks zu fördern. Dies erspart weitere Förderelemente wie beispielsweise Pumpen und kann das Antriebssystem vereinfachen. Da im Triebwerksbetrieb die Tanktemperatur bei der Gasentnahme aufgrund der Verdampfungsenthalpie der verdampfenden flüssigen Phase abfällt, kommt es zu einem Absinken des Brennkammerdrucks. Ist für einen bestimmten Betriebspunkt ein konstanter Brennkammerdruck gefordert, ist ein entsprechender Regelvorgang durch das Öffnen und Schließen von Regelventilen notwendig [5]. Das vorrangige Ziel dieser Arbeit ist es, die Regelung

des Triebwerks und des Brennkammerdrucks bei sich ändernden Strömungsbedingungen mittels RL basierter Regler zu gewährleisten. Um das Verhalten der neuronalen Netze optimal zu untersuchen und besser der KI zuordnen zu können, wird ein vereinfachtes Kaltgas-Prüfstandsmodell verwendet. Hierfür wird der Prüfstand lediglich mit Stickstoff durchströmt und auf eine Kombination aus mehreren Treibstoffkomponenten verzichtet.

2.2. Reinforcement Learning

Beim Reinforcement Learning handelt es sich um eine Methode des maschinellen Lernens. Die Anwendungen sind nicht rein auf die Technologiebranche beschränkt. Auch in der Medizin [18] und im Finanzwesen werden mittels RL-Algorithmen Vorhersagen zu bestimmten Sachverhalten getroffen. Besonders im Aktienhandel gewinnt die Methode aktuell an Popularität. Es konnte gezeigt werden, dass RL-Algorithmen erfolgreich Aktienkursprognosen anfertigen und Verkaufsentscheidungen treffen konnten, um finanzielle Gewinne zu optimieren [19]. Weitere Beispiele sind unter anderem in der Videospielbranche [20, 8], in der Robotik [21] oder in der Flugregelung [22] zu finden. Die Algorithmen des maschinellen Lernens können in die folgenden drei Kategorien eingeteilt werden [23]:

Überwachtes Lernen Zu den Aufgaben des überwachten Lernens zählt vor allem die Datenregression und die Klassifizierung von Datensätzen. Für das Training müssen diese Daten vorab entsprechend gekennzeichnet werden. Ein Beispiel aus der Bilderkennung ist die automatische Erkennung von Ampeln. Ein Algorithmus aus der Klasse des überwachten Lernens benötigt für das Training in diesem Fall eine Reihe von Bildern, auf denen Ampeln bereits gekennzeichnet wurden.

Unüberwachtes Lernen In dieser Klasse sind die Zielwerte nicht im voraus bekannt. Der Algorithmus versucht statt dessen, ein Muster in den Datensätzen zu erkennen. Hierfür bedient er sich der automatischen Segmentierung (Clustering) oder der Komprimierung von Daten zur Dimensionsreduktion. Im Gegensatz zum überwachten Lernen kann in einer Bildserie auf diese Weise beispielsweise zwar kein Tier identifiziert werden, allerdings kann ein entsprechender Algorithmus abgebildete Tiere unter den entsprechenden Umständen ihrer Gattung zuordnen.

Bestärkendes Lernen (Reinforcement Learning) Beim bestärkenden Lernen wird mittels eines Belohnungssystems, welches richtige Entscheidungen bestärkt, ein bestimmtes Verhalten angelernt. Dies geschieht ohne das Kennen der zugrundeliegenden mathematischen Zusammenhänge des Systems. Der Vorgang ist dem biologischen Lernprozess eines Menschen nachempfunden. Beispielsweise kann ein Kind ohne die Kenntnis der biomechanischen Gleichungen des menschlichen Bewegungsapparates das Laufen lernen.

2.2.1. Vorteile und Herausforderungen

Der Vorteil des Reinforcement Learnings besteht im Wesentlichen darin, dass ein einmalig trainierter Regler die Vorgabewerte mit minimalem Rechenaufwand bestimmen kann.

Dieser Sachverhalt ermöglicht eine Echtzeitregelung, was insbesondere für die Regelung von Systemen in der Luft- und Raumfahrt von großer Relevanz ist. Weiterhin können komplexe Regelaufgaben und Regelziele berücksichtigt werden, welche die Möglichkeiten konventioneller PID-Regler übersteigen [24]. Dazu zählen insbesondere Aufgaben, bei denen sehr viele Größen geregelt werden müssen oder sich einzelne Parameter über mehrere Größenordnungen ändern. Aufgrund dieser Eigenschaften ist die Regelung nichtlinearer Vorgänge wie das Regeln eines Raketentriebwerks mittels Methoden des Reinforcement Learnings im Interesse des Deutschen Zentrums für Luft- und Raumfahrt und das Thema dieser Arbeit [4].

Trotz der genannten Vorteile steht das Reinforcement Learning vor einigen Herausforderungen, welche sich hauptsächlich durch das übertragen der Algorithmen in Anwendungen der realen Welt ergeben[10, 25]. Diese sind in Tabelle 2.1 zusammengefasst.

Herausforderung	Beschreibung
Diskrepanz zwischen Simulation und Realität	Bei der Übertragung der im Training gewonnenen neuronalen Netze auf ein reales Modell treten selbst bei guter Kalibrierung Fehler auf, da Messdaten immer einem Rauschen unterliegen. Auch ist es praktisch nicht möglich, die Realität vollständig in der Simulation abzubilden. [11].
Begrenzte Erfahrungswerte	In realen Systemen stehen bedingt durch die Zeit und die charakteristische Geschwindigkeit nur eine begrenzte Anzahl an Erfahrungswerten zur Verfügung. Außerdem sind reale Tests mit relativ hohen Kosten verbunden. In Simulationen ist dagegen die Rechenleistung der limitierende Faktor.
Bereichseinschränkungen	Um Schäden vorzubeugen, dürfen während des Betriebs einer realen, sicherheitskritischen Struktur wie etwa einem Triebwerksprüfstand bestimmte Betriebsbereiche nicht angefahren werden. Diese Beschränkungen sollten vorab in der Simulation berücksichtigt werden.
Echtzeitfähigkeit	Für die Regelung eines realen Systems muss die Regelung ihre Aktionen innerhalb der Regelfrequenz in Echtzeit berechnen.
Offline Learning	Da für reale Systeme das Lernen durch Online-Interaktion mit dem System häufig zu teuer oder zu zeitintensiv ist, müssen Algorithmen entwickelt werden, die anhand von bereits gesammelten, gespeicherten Daten eine Policy erlernen.
Stabilität des Lernprozesses	Der Lernprozess beim Deep Reinforcement Learning kann instabil und stochastisch sein. Somit können keine exakten Voraussagen über das Verhalten während des Lernens getroffen werden. [25].

Tab. 2.1.: Tabelle mit verschiedenen Herausforderungen des Reinforcement Learnings

Aufgrund dieser Herausforderungen findet das Reinforcement Learning bisher hauptsächlich in Simulationsumgebungen statt. Eine gute Kenntnis der genannten Unterschiede zwischen Simulation und der realen Anwendung ist notwendig, um die trainierten neuronalen Netze erfolgreich zu übertragen. Hierfür ist es hilfreich, sich weiterer Methoden zu bedienen, um die Robustheit des RL-Reglers zu erhöhen. Ein Beispiel hierfür ist die in Kapitel 2.4 näher ausgeführte *Domain Randomization* oder das Einbringen von Rauschen bestimmter physikalischer oder konstruktiver Größen in das Training der neuronalen Netze.

2.2.2. Anschauliche Beschreibung

Der Ablauf eines RL basierten Lernprozesses lässt sich grundsätzlich wie in Bild 2.1 darstellen. Reinforcement Learning arbeitet in diskreten Zeitschritten [26]. In jedem Zeitschritt werden von der Umgebung (Environment) Informationen über den aktuellen Systemzustand an den Regler geschickt. Der Systemzustand ist in der Abbildung mit s_t gekennzeichnet. Der Regler, im Fachjargon Agent genannt, berechnet daraufhin mit einer Entscheidungsregel und den aktuellen Zustandsdaten s_t eine Handlungsanweisung (a_t). Die Handlungsanweisung wird als Aktion an die Umgebung gesendet. Hinzu kommt eine Bewertung über den aktuellen Zustand. Dafür wird eine Belohnung oder Bestrafung (r_t, Reward) vergeben. Das Ziel des Reglers ist es, die Summe aller Rewards einer Episode zu maximieren. Eine Episode kann eine festgelegte Anzahl an Zeitschritten sein, bevor der Vorgang wiederholt wird. Die Entscheidungsregel des Reglers wird dazu derart optimiert, dass die Bewertung des zukünftigen Systemzustands maximiert wird.

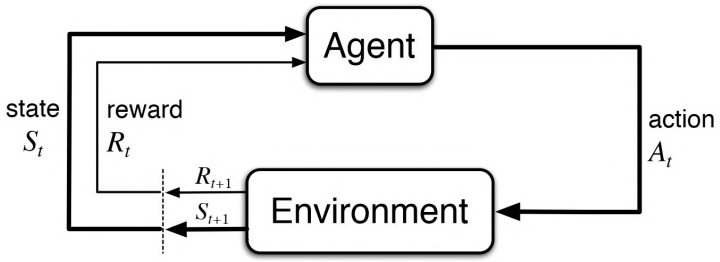


Abbildung 2.1.: Grundsätzlicher Ablauf eines RL Lernprozesses aus [26]

Um im Verlauf der Arbeit den entsprechenden Begrifflichkeiten des Reinforcement Learnings besser folgen zu können, werden hier die wesentlichen Begriffe in ihrer englischen Bezeichnung aufgelistet und beschrieben [5]:

Agent Bezeichnung für den Algorithmus beziehungsweise den Regler, der anhand der Entscheidungsregel die Handlungen berechnet.

Environment Umgebung, in welcher der Algorithmus arbeitet. In der Regelungstechnik ist dies die Regelstrecke. Die Umgebung kann ein Videospiel, ein Roboterarm

oder ein Antriebssystem sein. Der Agent kann durch seine Handlungsanweisungen beziehungsweise Aktionen (a_t) mit der Umgebung interagieren.

State Systemzustand der Umgebung und des Agenten, zum Beispiel Position und Geschwindigkeit eines Regelventils.

Reward Eine skalare Größe, die als Maßstab dafür gesehen werden kann, ob ein Zustand der Umgebung als gut oder schlecht bewertet wird. Die Berechnungsfunktion für den Reward muss für jede Anwendung individuell angepasst werden.

Action Handlungsanweisung des Agenten an die Umgebung und somit eine Sollwertvorgabe einer Stellgröße. Diejenigen Größen, die der Algorithmus beeinflussen kann werden dabei als *Action-Space* bezeichnet.

Observation-Space Im Beobachtungsraum befinden sich diejenigen Größen, die dem Algorithmus als Eingangsdaten dienen. Sie beschreiben den Systemzustand und können auch individuell festgelegt werden.

Policy Als Entscheidungsregel des Agenten ist die Policy eine Funktion, die Zustände s auf Aktionen a abbildet. Die Policy bestimmt, wie der Agent auf verschiedene Systemzustände reagiert, indem eine Auswertung mit den Werten des State stattfindet. Die Policy kann eine Tabelle, eine Funktion oder ein neuronales Netz sein. Sie kann stochastisch oder deterministisch sein.

Die Zusammenhänge der oben beschriebenen Begriffe kann anhand eines orbitalen Raketenantriebssystems anschaulich beschrieben werden. Soll beispielsweise der Brennkammerdruck auf einen stationären Wert geregelt werden, muss der Reinforcement Learning Agent die Möglichkeit haben, die Ventilposition für das Regelventil mittels eines Stellsignals einstellen zu können. Dies entspricht der Handlungsanweisung, beziehungsweise der *Action*. Die Bewertung des erreichten Ergebnisses kann anhand der Abweichung vom gewünschten Brennkammerdruck erfolgen. Hierfür kann eine *Rewardfunktion* definiert werden, die den Fehler vom gewünschten Brennkammerdruck beinhaltet. Das Ergebnis dieser Funktion ist entweder ein positiver Zuschlag für eine Verbesserung oder eine Strafe in Form eines negativen Werts. Damit die Handlungsanweisungen sinnvoll erfolgen, muss der *Observation-Space* geeignete Größen enthalten. Dies können beispielsweise der Brennkammerdruck, die Ventilposition und der Fehler zum Solldruck sein. Die Vorgehensweise für die in dieser Arbeit untersuchten Modelle wird in Kapitel 3 nochmals im Detail erläutert.

2.2.3. Mathematische Formulierung

Als grundlegende Formulierung des Reinforcement Learnings gilt der Markowsche Entscheidungsprozess (engl. *Markov Decision Process*, MDP) [27]. Ein MDP ist ein Modell eines Entscheidungsproblems und besteht aus einem Tupel $(s,a,\mathcal{P},r,\gamma)$ mit folgender Notation:

s	$\widehat{=}$	State, ein Zustand aus dem Raum aller möglichen Zustände S
a	$\widehat{=}$	Action, eine Aktion aus dem Raum aller möglichen Aktionen A
$\mathcal{P}$	$\widehat{=}$	Zustandsübergangsfunktion
r	$\widehat{=}$	Reward
γ	$\widehat{=}$	Discount-Faktor

Zu einem Zeitpunkt t befindet sich das betrachtete System in einem Zustand s_t, der RL-Algorithmus wählt eine Aktion a_t aus und führt das System im nächsten Zeitschritt in den Zustand s_{t+1} über. Zu diesem Zeitschritt gehört der Reward r_t. Dabei hängt der Zustand s_{t+1} nur vom Zustand s_t und der Aktion a_t ab. Somit spielt es für den Übergang von t nach $t+1$ keine Rolle, was in Zustand s_{t-1} passiert ist. Diese Eigenschaft wird Markow-Eigenschaft genannt [26]. Allerdings ist dies eine idealisierte Betrachtung, welche in der Praxis des Reinforcement Learnings nicht immer zutrifft.

Die Zustandsübergangsfunktion $\mathcal{P}$ gibt die Wahrscheinlichkeit eines Übergangs zu s_{t+1} an, wenn vom Zustand s_t die Aktion a_t gewählt wurde. $\mathcal{P}$ gilt als die dynamischen Eigenschaften des Systems, welches sowohl deterministisch oder stochastisch sein kann [26]. Die Funktion $\mathcal{P}$ gibt entsprechend der physikalischen Eigenschaften der Umgebung den nächsten Systemzustand oder eine Wahrscheinlichkeitsverteilung der darauffolgenden Zustände an.

Eine Aneinanderreihung mehrerer Markowscher Entscheidungsprozesse wird Trajektorie beziehungsweise Episode genannt [27]. Sie lässt sich als Abfolge von Aktionen und Zuständen wie folgt schreiben:

$$\tau = (s_0, a_0, s_1, a_1, ...) \tag{2.1}$$

Der Ablauf einer Trajektorie ist in Abbildung 2.2 dargestellt. Das System befindet sich zunächst in Zustand s_0. Eine Aktion a_0 wird ausgeführt und anschließend der Reward r_1 bestimmt. Das System befindet sich danach im neuen Zustand s_1. Dieses Schema wird wiederholt, bis der Endzustand erreicht ist.

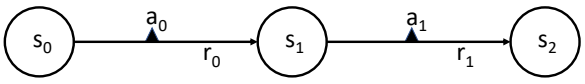


Abbildung 2.2.: Veranschaulichung einer Trajektorie innerhalb eines MDP

Je nachdem, wie die Aufgabe formuliert ist, können Episoden endlich oder unendlich sein. Für endliche Episoden muss der Endpunkt definiert werden. Ein Beispiel für eine unendliche Episode ist das möglichst lange Ausbalancieren eines Stabes. Für das orbitale Raketenantriebssystem definiert sich eine Episode als der Zeitraum zwischen der erstmaligen Öffnung des Ventils und dem Abschalten des Triebwerks.

Werden die einzelnen Werte des Rewards zu jedem Schritt der Episode aufsummiert, ergibt sich der summierte Reward, *Return* $G(\tau)$ genannt. Für eine unendliche Episode muss gewährleistet werden, dass der aufsummierte Reward nicht gegen unendlich strebt.

Hierfür wird ein *Discount-Faktor* γ eingeführt, wobei $0 < \gamma < 1$. Der Discount-Faktor kann auch bei endlichen Episoden genutzt werden. Damit wird erzielt, dass weit in der Zukunft liegende Ergebnisse nicht so stark gewertet werden. Das Ergebnis aus Gleichung 2.2 wird auch *discounted Return* genannt.

$$G(\tau) = \sum_{t=0}^{\infty} \gamma^t r(s_t) \tag{2.2}$$

Ziel des RL-Agenten ist es, unter Variation der Entscheidungsregel (*Policy*), den erwarteten Return zu maximieren. Die Policy π ist somit die Lösung des MDP [27].

2.2.4. Value-Funktionen und Bellman-Gleichung

Eine unter RL-Algorithmen weit verbreitete Idee ist die Nutzung der *Value-Funktion* V. Diese gibt an, wie zielführend es für den Agenten ist, sich in einem bestimmten Zustand zu befinden. Als Maß dafür dienen die zukünftigen Belohnungen, die innerhalb einer Episode erwartet werden. Diese hängen von den jeweiligen Aktionen des Agenten ab, weshalb Value-Funktionen V_π in Abhängigkeit von der Entscheidungsregel (Policy) π definiert werden [26]:

$$V_\pi(s) = \mathbb{E}_\pi[G_t | s = s_t] = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r(s_t) | s = s_t \right] \tag{2.3}$$

Die Value-Funktion gibt somit den erwarteten Return an, wenn sich das System im Zustand s_t befindet und die Aktionen gemäß der Policy π vorgenommen werden [27]. Da die Zustandsübergangsfunktion P zufällig sein kann, wird in die Notation der Erwartungswert $\mathbb{E}$ einbezogen. Neben der Value-Funktion V kann weiterhin die *Action-Value-Funktion* Q wie folgt definiert werden:

$$Q_\pi(s, a) = \mathbb{E}_\pi[G_t | s = s_t, a = a_t] = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r(s_t) | s = s_t, a = a_t \right] \tag{2.4}$$

Sie gibt den zu erwartenden Reward an, falls im Zustand s_t zunächst eine Aktion a_t gewählt wird und anschließend immer die von der Entscheidungsregel π vorhergesagten Aktionen ausgeführt werden.
Mit den bisher erläuterten Begriffen der Value- und Action-Value-Funktion kann die optimale Policy π^* folgendermaßen angegeben werden: Eine Policy π ist besser oder gleich gut wie eine andere Policy π', wenn ihr erwarteter Return für alle Zustände größer oder gleich dem von π' ist [26].

Die optimale Value-Funktion $V^*(s)$ und Action-Value-Funktion $Q^*(s)$ lautet unter Berücksichtigung der optimalen Policy π^* folgendermaßen:

$$V^*(s) = \max_\pi V_\pi(s) \tag{2.5}$$

$$Q^*(s, a) = \max_\pi Q_\pi(s) \tag{2.6}$$

Aus der Definition von V und Q folgt ein weiterer Zusammenhang der beiden Funktionen welcher in Gleichung 2.7 beschrieben ist. Die Schreibweise $\max_a$ kennzeichnet hier die Vorgehensweise, durch welche diejenige Aktion a_t gewählt wird, die den maximalen Reward ergibt. Die optimale Policy π^* wählt ebenfalls die Aktion a_t aus.

$$V^*(s) = \max_a Q^*(s,a) \tag{2.7}$$

Auf Grundlage der Value-Funktionen kann eine rekursive Beziehung aufgestellt werden, welche beim Reinforcement Learning verwendet wird [26].

$$V_\pi(s) = \mathbb{E}_\pi[r(s_t,a_t) + \gamma V_\pi(s_{t+1})] \tag{2.8}$$

Sie lässt sich analog auch für die Action-Value-Funktion Q_π definieren:

$$Q_\pi(s,a) = \mathbb{E}_\pi[r(s_t,a_t) + \gamma \mathbb{E}[Q_\pi(s_{t+1}, a_{t+1})]] \tag{2.9}$$

Gleichung 2.8 ist die Bellman-Gleichung für V_π. Sie drückt die Beziehung zwischen dem Wert eines Zustandes $V_\pi(s)$ und dem Wert des darauffolgenden Zustandes $V_\pi(s_{t+1})$ aus [26]. Ihre rekursive Definition bringt mit sich, dass die Funktion V_π als Funktion von sich selbst ausgedrückt wird.

Das Optimalitätsprinzip von Bellman besagt, dass sich bei einigen Optimierungsproblemen jede Optimallösung aus optimalen Teillösungen zusammensetzt. Für das Reinforcement Learning bedeutet dies, dass eine optimale Lösung nur erreicht werden kann, wenn jede Aktion auch eine optimale Aktion ist. Entsprechend ist dies in der Formulierung der Bellman-Gleichung zu finden. Der Wert des Zustandes s_t entspricht dem Wert des aktuellen Rewards $r(s_t, a_t)$ addiert mit dem Wert aller zukünftigen Zustände $V_\pi(s_{t+1})$. Dabei wird der Wert der zukünftigen Zustände mit dem Discount-Faktor γ gewichtet [28].

2.3. RL-Algorithmen

Da eine schnell wachsende Anzahl verschiedener RL-Algorithmen existiert, ist es sinnvoll, sich zunächst die kategorische Strukturierung der Verfahren zu verdeutlichen. Eine mögliche Kategorisierung ist in Abbildung 2.3 dargestellt. Als Kriterium für die Zuordnung dient die zum Teil deutlich unterschiedliche Optimierung der Policy der einzelnen Algorithmen. Auf die wesentlichen Aspekte soll in diesem Abschnitt eingegangen werden. Anschließend erfolgt eine genauere Erläuterung der in dieser Arbeit eingesetzten Algorithmen.

Abbildung 2.3 zeigt zunächst eine Unterteilung der Algorithmen anhand der Notwendigkeit eines Modells, welches zur Vorausplanung für den Agenten herangezogen wird. **Modellbasierte Algorithmen** greifen auf Funktionen zurück, welche die Zustandsübergänge und in manchen Fällen den Reward eines Systems voraussagen. Die Modelle können hierfür entweder gelernt oder vorgegeben werden. Ein Vorteil modellbasierter Algorithmen ist die Probeneffizienz. Es werden weniger Trainingsdaten als bei

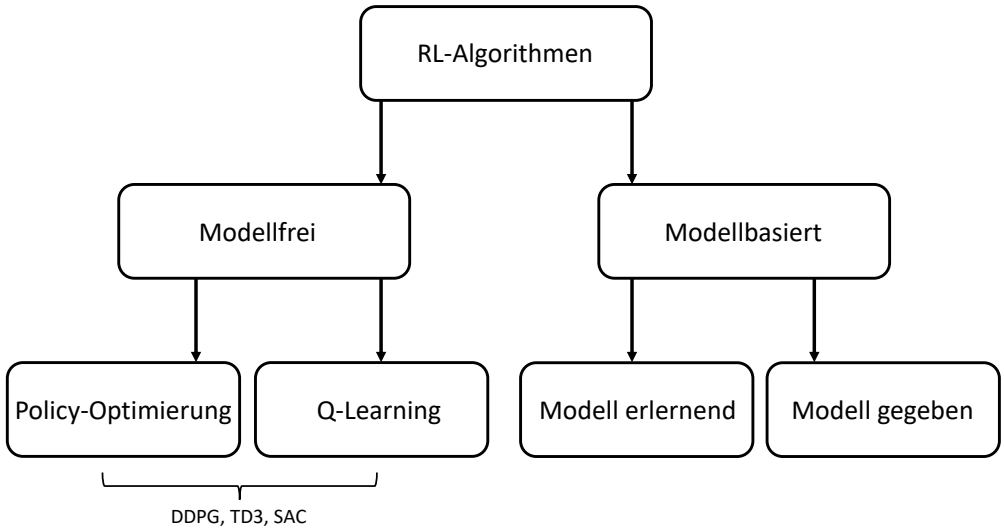


Abbildung 2.3.: Kategorisierung von Reinforcement Learning Algorithmen nach [29]

modellfreien Verfahren benötigt [30]. Allerdings besteht die Gefahr, dass das Modell fehlerhaft ist oder Ungenauigkeiten aufweist und somit das reale Modell unzulänglich widerspiegelt. Außerdem entsteht bei dieser Verfahrensklasse ein zusätzlicher Aufwand, da zunächst ein geeignetes Modell gefunden und implementiert werden muss. Hierin liegt ein Vorteil der **modellfreien Algorithmen**. Sie sind einfacher zu implementieren und stehen häufig in *Open Source*-Projekten zur freien Verfügung [31]. Ein Modell, welches das Systemverhalten beschreibt, wird bei dieser Klasse von Verfahren nicht benötigt. Da in der vorliegenden Arbeit modellfreie Algorithmen eingesetzt werden, folgt eine vertiefte Behandlung ebendieser.

Modellfreie Verfahren unterscheiden sich dadurch, welche Funktion sie erlernen. Es gibt Ansätze, bei denen die Policy π direkt gelernt wird. Andere erlernen zuerst die Value-Funktion V oder die Action-Value-Funktion Q, um daraus auf die Policy zu schließen. Bei den **Policy-basierten Ansätzen** kann die Policy direkt als Funktion $\pi_\Theta(a|s)$ mit den Parametern Θ dargestellt werden. Sie bestimmt die zu den jeweiligen Zuständen s passende Aktion a. Ziel ist es, die Parameter Θ zu optimieren. Beim *Deep Reinforcement Learning* besteht die Policy aus einem neuronalen Netz, dessen Gewichte über den Parameter Θ einfließen. Das hierfür aufgestellte Gradientenverfahren lautet:

$$\Theta_{k+1} = \Theta_k + \beta \nabla_\Theta J(\Theta) \tag{2.10}$$

Hier kennzeichnet β die Schrittgröße und $J(\Theta)$ den erwarteten Reward einer Episode bei Befolgung der Policy. $J(\Theta)$ kann als skalarer Maßstab für die Bewertung der Policy gesehen werden [12]. $\nabla_\Theta J(\Theta)$ ist damit der Gradient des Bewertungsmaßstabes bezüglich dessen Parametern. Bei einer Bewegung in Richtung dieses Gradienten finden sich die

Parameter, welche die maximale Bewertung ergeben. Mit dem *Policy-Gradient-Theorem* [26] kann die analytische Gleichung 2.11 zur Berechnung von $\nabla_\Theta J(\Theta)$ angegeben werden. Dies ermöglicht die Formulierung von *Policy-Gradient-Algorithmen.*

$$\nabla_\Theta J(\pi_\Theta) = \mathbb{E}_\pi \left[\sum_{t=0}^{T} \nabla_\Theta log \pi_\Theta(a_t|s_t) G(\tau) \right] \tag{2.11}$$

Der Erwartungswert kann durch den Mittelwert aus einer ausreichenden Anzahl von Trainingsdaten bestimmt werden. Wenn $\mathcal{F} = \tau_{i=1,N}$ ein Set an Episoden τ bezeichnet, ergibt sich eine berechenbare Gleichung des Policy-Gradienten [29]:

$$\nabla_\Theta J(\pi_\Theta) = \frac{1}{|\mathcal{F}|} \sum_{\tau \in \mathcal{F}} \sum_{t=0}^{T} \nabla_\Theta log \pi_\Theta(a_t|s_t) G(\tau) \tag{2.12}$$

Verfahren, welche sich diesen Ansätzen bedienen heißen **on-Policy-Algorithmen**. Für das Training verwenden sie nur diejenigen Erfahrungsdaten, die mit Handlungsanweisungen gemäß der neusten Version der Policy erzeugt wurden. Ältere Daten werden verworfen, was die Probeneffizienz dieser Verfahrensklasse senkt, da bei jeder Iteration der Policy komplett neue Trainingsdaten erzeugt werden müssen.

Im Gegensatz dazu erlernen **off-Policy-Algorithmen** anstatt der Policy eine Approximation der Action-Value-Funktion Q. Sie verwenden alle Erfahrungswerte, die im Laufe des Trainings gesammelt werden [26]. Die Approximation der optimalen Action-Value-Funktion $Q_\pi(s,a)$ findet über eine als Zielfunktion formulierte Bellman-Gleichung statt [5]. Der dem Prinzip nach benannte Q-Learning-Algorithmus ist ein Beispiel für ein Verfahren, welches die Q-Funktion lernt [32].

Darüber hinaus gibt es eine ganze Klasse an Algorithmen, welche sich beide Ansätze zunutze machen, indem Policy Optimierung und Q-Learning miteinander kombiniert werden. Dazu gehören der DDPG-Algorithmus [33] und dessen erweiterte Versionen TD3 [34] und SAC [35]. Die Q-Funktion (*Critic*) wird hier genutzt, um die Policy (*Actor*) zu verbessern. Diese Verfahren werden demnach **Actor-Critic-Algorithmen** genannt.

2.3.1. (Deep) Q-Learning

Ein grundlegenes Verfahren, welches als Baustein für eine Reihe an fortschrittlicheren Algorithmen dient, ist der *Q-Learning*-Algorithmus. Dieser verwendet eine Tabelle, in der alle Zustände und die jeweiligen möglichen Aktionen aufgelistet sind. Aufgrund des tabellarischen Aufbaus kann er nur in diskreten Aktionsräumen angewendet werden. Zu jedem Paar aus State und Action gibt es einen zugehörigen Q-Wert, welcher mit folgender Formulierung der Bellman-Gleichung iterativ berechnet werden kann [32]:

$$Q^{neu}(s_t, a_t) \leftarrow Q^{alt}(s_t, a_t) + \beta \cdot (r_t + \gamma * \underset{a}{max} Q(s_{t+1}, a_t) - Q^{alt}(s_t, a_t)) \tag{2.13}$$

Die neuen Q-Werte setzen sich aus dem alten Q-Wert und einem Anteil, der sich aus dem Reward des aktuellen Zustandes und dem maximal im nächsten Schritt möglichen Q-Wert ergibt, zusammen. Der Discount-Faktor γ ist bekannt. Die Lernrate β beeinflusst,

wie stark die Ergebnisse des nächsten Schrittes berücksichtigt werden. Ist $\beta = 0$, so kann der Algorithmus keine Informationen aus den nachfolgenden Schritten ziehen. Wird $\beta = 1$ gesetzt, so werden die alten Q-Werte verworfen. Als nächstes wird diejenige Aktion gewählt, welche den größten Q-Wert verspricht [5, 32].

Algorithmus 1 Q-Learning

1: Initialisiere die Q-Tabelle für alle möglichen Zustände S und Aktionen A
2: **for** $t = 1 \rightarrow T$ **do** ▷ Berechnung einer Episode
3: Wähle eine Aktion a_t anhand der Policy, die aus Gleichung 2.7 bestimmt wird.
4: Führe die Aktion aus und beobachte r_t und s_{t+1}.
5: Update der Q-Funktion gemäß:
6: $$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \beta \left[r + \gamma \underset{a}{max} Q(s_{t+1},a_{t+1}) - Q(s_t,a_t) \right]$$
7: $s_t \leftarrow s_{t+1}$

Der Ablauf des Verfahrens für eine Episode ist in Pseudocode 1 schematisch zusammengefasst [32]. Im ersten Schritt werden alle möglichen Zustände S, alle Aktionen A und die zugehörigen Rewards R als Matrix definiert. Die Q-Wert-Matrix wird dabei zunächst mit 0 initialisiert. Die Reward-Matrix beinhaltet die Verteilung des Rewards auf die jeweiligen Zustände. Nach der Initialisierung erfolgt eine Schleife zur iterativen Bestimmung der Q-Werte. Anschließend findet die Auswahl einer Aktion entweder zufällig oder anhand einer vorher festgelegten Erkundungsstrategie statt (vgl. Abschnitt 2.3.2). Durch die teilweise zufällige Auswahl an Aktionen wird gewährleistet, dass immer neue Bereiche der Umgebung erkundet werden und dass der Agent sich nicht in einem lokalen Optimum festsetzt. Die Aktion a wird ausgeführt und dabei der neue Zustand s' und Reward r beobachtet. Mit s' und r können die Q-Werte entsprechend Gleichung 2.13 erneut berechnet werden. Diese Schritte werden wiederholt, bis der Zustand s dem finalen Zustand entspricht. Daraufhin beginnt die nächste Episode. Dieser Vorgang dauert so lange an, bis die Q-Werte konvergieren.

Aufgrund der Formulierung der Q-Werte als Tabelle ist nur ein diskretes Set an Zuständen möglich. Die Verteilung der Zustände wie beispielsweise Druck und Massenstrom in einem Raketentriebwerk sind allerdings kontinuierlich. Ebenso sind die möglichen Aktionen eine kontinuierliche Größe, da Regelventile alle Positionen zwischen vollständig geöffnet und komplett geschlossen anfahren können [5]. Wird eine Diskretisierung der Action-Space-Größen durchgeführt, so ergeben sich aufwändig zu lösende, hochdimensionale Q-Tabellen. Bei kontinuierlichen Q-Funktionen besteht eine Schwierigkeit darin, den maximalen Q-Wert aller Aktionen zu finden. Im diskreten Raum ist dies durch den Vergleich aller Aktionen möglich, im kontinuierlichen Fall ist hierfür ein iterativer Optimierungsprozess in jedem Zeitschritt notwendig [33]. Für die Regelung eines Raketentriebwerks ist der Q-Learning-Algorithmus somit nicht praktikabel. Als erfolgreiche Alternative zur nichtlinearen Funktionsapproximation haben sich in den letzten Jahren neuronale Netze bewährt [36].

Eine entsprechende Lösung der eben beschriebenen Problematik bietet das Deep Reinforcement Learning, in welchem die Q-Tabelle durch ein neuronales Netz ersetzt wird.

Grundlegend kann dabei wie beim Q-Learning-Algorithmus vorgegangen werden. Um Instabilitäten beim Training vorzubeugen und die Konvergenz der Verfahren zu gewährleisten, wurden diese Algorithmen weiter entwickelt [33]. Ein Ergebnis dessen ist der **Deep-Q-Learning** Algorithmus. Um Instabilitäten zu verhindern, werden beim Deep-Q-Learning sogenannte *Replay-Buffer* eingesetzt. Die Erfahrung des Agenten zu jedem Zeitschritt $e_t = (s_t, a_t, r_t, s_{t+1})$ wird in einem Speicher abgelegt. Im Training werden daraufhin bei jeder Iteration zufällige Erfahrungen e aus der Menge an gespeicherten Transitionen ausgewählt. Dadurch wird die Korrelation der Erfahrungen einer Episode unterbrochen, welche als Grund für die Instabilitäten gilt. Darüber hinaus kann damit jede Transition e für mehrere Iterationen genutzt werden, was zu einer höheren Dateneffizienz führt [20].

2.3.2. Actor-Critic-Methoden

Um die Vorteile der Policy-Gradient und Q-Learning basierten Algorithmen nutzen zu können, ist es möglich, beide Verfahren zu kombinieren. Damit wird erreicht, dass off-Policy-Algorithmen mit einem kontinuierlichen Action- und Observation-Space gebildet werden können. Diese Methoden werden Actor-Critic-Algorithmen genannt. Der grundlegende schematische Ablauf eines solchen Verfahrens ist in Abbildung 2.4 dargestellt.

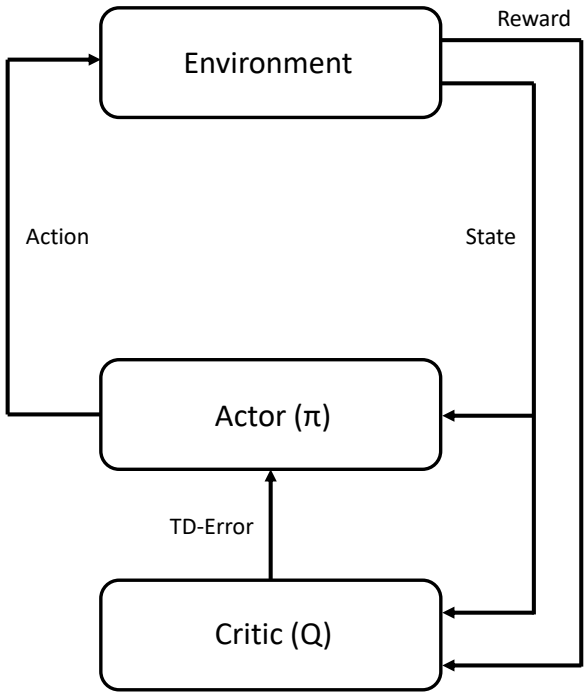


Abbildung 2.4.: Graphische Darstellung eines generischen Actor-Critic-Algorithmus nach [37]

Es ist ersichtlich, dass sowohl die Policy π_Θ als auch die Action-Value-Funktion Q gleichzeitig gelernt werden. Weiterhin lässt sich der Erwartungswert in der Actor-Critic-Formulierung aufteilen (vgl.Gleichung 2.11):

$$\nabla_\Theta J(\pi_\Theta) = \mathbb{E}_\pi \left[\sum_{t=0}^{T} \nabla_\Theta log \pi_\Theta(a_t|s_t) \right] \mathbb{E}_\pi \left[G(\tau) \right] \tag{2.14}$$

Der zweite Term in Gleichung 2.14 entspricht laut Gleichung 2.4 der Q-Funktion.

$$\mathbb{E}\left[G(\tau) \right] = Q(s_t,a_t) \tag{2.15}$$

Der Policy-Gradient kann also mit der Q-Funktion als *Critic* berechnet werden. Die Policy entscheidet, welche Aktion getätigt wird. Nach jeder Auswahl einer Aktion bewertet der Kritiker den neuen Zustand, um festzustellen, ob die Dinge besser oder schlechter als erwartet gelaufen sind. Hierfür wird der *Time Difference Error* (TD-Fehler) berechnet. Ist der TD-Fehler positiv, deutet dies darauf hin, dass die Tendenz, die aktuelle Aktion zu wählen, in Zukunft verstärkt werden sollte. Ein negativer TD-Fehler bringt zum Ausdruck, diese Tendenz möglichst abzuschwächen [37].

Viele der gebräuchlichen Actor-Kritik-Verfahren sind aufeinander aufgebaut. Daher sollen zunächst zwei Algorithmen vorgestellt werden, die dem in dieser Arbeit verwendeten **Soft Actor-Critic-Algorithmus (SAC)** zugrunde liegen. Dabei werden Prinzipien, die wiederholt Anwendung finden, genauer erläutert.

Deep Deterministic Policy Gradient

Als erstes Beispiel für ein Actor-Critic-Verfahren wird der DDPG-Algorithmus (Deep Deterministic Policy Gradient) eingeführt und die dort auftretenden Maßnahmen *Replay Buffer* und *Separates Target Network* vorgestellt.

Die Approximation für die optimale Entscheidungsregel π_Θ und die optimale Q-Funktion $Q^*(s,a)$ werden beim DDPG gleichzeitig erlernt. Letztere dient der Policy als Kritiker. Die Approximation an Q^* muss bekanntermaßen die Bellman-Gleichung 2.9 erfüllen. $Q(s,a)$ wird durch ein neuronales Netz mit den Gewichten Φ angenähert. Hierfür lässt sich eine Fehlergleichung aufstellen die anzeigt, wie weit die Approximation Q_Φ davon entfernt ist, der Bellman-Gleichung zu genügen [33]:

$$L(\Phi, \mathcal{D}) = \underset{(s_t,a_t,r_t,s_{t+1}) \ \mathcal{D}}{\mathbb{E}} \left[\left(Q_\Phi(s_t,a_t) - y \right)^2 \right] \tag{2.16}$$

Wird die Funktion $L(\Theta, D)$ minimiert, so entspricht die Approximation Q_Φ einer Lösung der Bellman-Gleichung und somit der optimalen Q-Funktion. $\mathcal{D}$ beinhaltet hier eine Reihe an Übergängen (s_t,a_t,r_t,s_{t+1}). Die Parameter Θ der Policy werden gemäß Gleichung 2.10 berechnet. Die Größe $\nabla_\Theta J$ kann mit dem deterministischen Policy-Gradienten bestimmt werden [38]. y wird Zielfunktion genannt, da die Q-Funktion diesem Wert entsprechen soll. Diese kann wie folgt angegeben werden:

$$y = r + \gamma \underset{a_{t+1}}{max} Q_\Phi(s_{t+1},a_{t+1}) \tag{2.17}$$

Aufgrund der Verwendung nichtlinearer Funktionen zur Approximation ist die Konvergenz des Algorithmus nicht mehr gesichert. Auch hier können ähnlich zum Deep-Q-Learning Maßnahmen eingeführt werden, um die Konvergenz des Verfahrens zu verbessern [5].

Replay Buffer Um die Funktionsweise des Optimierungsalgorithmus zu gewährleisten, müssen die Transitionen voneinander unabhängig und gleichmäßig verteilt sein. Da die Transitionen innerhalb einer Episode nicht unabhängig sind, wird ein Zwischenspeicher (*Buffer*) eingeführt, der zunächst alle Trainingsdaten speichert und in zufälliger Reihenfolge für das Training nutzbar macht.

Separates Target Network Da die Formulierung zur Approximation der Q-Funktion auch in der Zielfunktion vorkommt, neigt die Minimierung von 2.16 dazu, instabil zu werden. Um dies zu vermeiden, wird das Ziel mit einem separaten Netzwerk berechnet. Die Parameter des Ziel-Netzes Θ' werden dabei in ausreichend kleinen Schritten („*soft target update*“) an das neu gelernte Netzwerk mit den Parametern Θ angepasst.

Twin Delayed DDPG

Da der DDPG Algorithmus häufig zu hohe Werte für die Q-Funktion annimmt, führt dies zu ineffizienten Policies. Der TD3-Algorithmus (Twin Delayed DDPG) als eine direkte Weiterentwicklung des DDPG-Algorithmus beinhaltet folgende drei Mechanismen, um diesen Effekt zu vermeiden [34]:

Clipped Double Q-Learning Es werden simultan zwei Q-Funktionen erlernt, wobei der kleinere Wert anschließend genutzt wird, um die Zielfunktion zu approximieren. Dadurch wird die Überhöhung der Q-Werte verringert.

Delayed Policy Updates Diese Idee beruht darauf, dass die Policy seltener neu berechnet wird, als die Q-Funktion. Das verspätete Policy-Update wird auf der Basis von Q-Werten mit geringerer Varianz vorgenommen. Da die Q-Werte als Kritik für die Berechnung des Policy-Updates dienen, wird ihre Qualität verbessert.

Target Policy Smoothing Beim DDPG-Algorithmus kommt es vor, dass die Q-Funktion für gewisse Aktionen einen lokalen Maximalwert erreicht und dadurch ein falsches Verhalten erlernt wird. Dies kann durch eine Glättung der Q-Funktion verhindert werden. Das Glätten wird über ein Rauschen in der Zielaktion bewerkstelligt.

Soft Actor-Critic

Der in dieser Arbeit verwendete Soft Actor-Critic (SAC) ist ein Algorithmus, der eine stochastische Policy auf eine off-Policy-Weise optimiert und somit eine Kombination aus stochastischer Policy-Optimierung und DDPG-artigen Ansätzen bildet. Er ist zwar kein direkter Nachfolger des TD3-Verfahrens, verwendet jedoch sowohl den *Clipped Double-Q-Trick* als auch das *Target Policy Smoothing* [39].

Ein zentrales Merkmal von SAC ist die sogenannte Entropie-Regularisierung. Die Policy wird so trainiert, dass ein Kompromiss zwischen dem erwarteten Reward und der Entropie, einem Maß für die Zufälligkeit der Policy, eingegangen wird. Eine höhere Entropie führt zu mehr *Exploration*, wodurch das spätere Lernen beschleunigt werden kann. Außerdem wird so verhindert, dass die Strategie vorzeitig zu einem schlechten lokalen Optimum konvergiert [35]. Im Kontext des Reinforcement Learnings versteht man unter dem Begriff der *Exploration* den Versuch, durch die Auswahl suboptimaler Aktionen neue Merkmale des Systems zu entdecken. Im Gegensatz dazu wird der Begriff der *Exploitation* als ein „gieriges“ (eng. *greedy*) Vorgehen definiert, bei dem die Agenten versuchen, einen höheren Reward zu erhalten, indem sie einen optimistisch geschätzten Wert anstatt des tatsächlichen Werts verwenden. Dies kann zu einer schnelleren Konvergenz des Problems führen.

Die Entropie ist in diesem Fall eine Größe, die angibt, wie zufällig eine Zufallsvariable ist. Wenn beispielsweise eine Münze so gewichtet ist, dass sie fast immer Kopf zeigt, hat sie eine niedrige Entropie. Wenn sie gleichmäßig gewichtet ist und dieselbe Chance für beide Ergebnisse hat, spricht man von einer hohen Entropie. Beim Entropie-regulierten Reinforcement Learning erhält der Agent bei jedem Zeitschritt einen Bonus-Reward, welcher proportional zur Entropie der Strategie bei diesem Zeitschritt ist [35].

Für die mathematische Beschreibung sei x eine Zufallsvariable mit der Wahrscheinlichkeitsdichtefunktion P. Die Entropie H wird damit wie folgt berechnet:

$$H(P) = \underset{x\ P}{\mathbb{E}}[-\log P(x)] \tag{2.18}$$

Anders als beim TD3-Verfahren stammen die in der Zielfunktion verwendeten Aktionen für den nächsten Zustand aus der aktuellen Policy anstatt aus der Target-Policy. Außerdem enthält die Zielfunktion einen zusätzlichen Term, der aus der Verwendung der Entropie-Regularisierung stammt [35]:

$$y = r + \gamma \underset{i=1,2}{min} Q_{\Phi'_i}(s', \bar{a'}) - \alpha log \pi_\Theta(\bar{a'}|s') \tag{2.19}$$

Der Entropie-Regulierungskoeffizient $\alpha > 0$ steuert explizit den Kompromiss zwischen Exploration und Exploitation, wobei ein höherer α-Wert einer stärkeren Exploration und ein niedrigerer α-Wert einer stärkeren Exploitation entspricht. Der optimale Koeffizient, welcher zum stabilsten Lernen mit der höchsten Belohnung führt, kann von Umgebung zu Umgebung variieren und erfordert eine sorgfältige Kalibrierung.

Die Policy sollte in jedem Zustand so handeln, dass der erwartete zukünftige Reward plus die erwartete zukünftige Entropie maximiert wird. Bei der Optimierung der Policy wird der *Trick der Reparametrisierung* angewandt, bei dem eine Stichprobe $\tilde{a}_\Theta(s)$ aus $\pi_\theta(s)$ gezogen wird. Dies geschieht durch Berechnung einer deterministischen Funktion des Zustands und der Policy-Parameter [39]. Die Policy wird optimiert nach [35]:

$$\nabla_\theta = \frac{1}{|\mathcal{B}|} \sum_{s \in \mathcal{B}} (\underset{i=1,2}{min} Q_{\Phi_i}(s, \tilde{a}_\Theta(s)) - \alpha log \pi_\Theta(\tilde{a}_\Theta(s)|s)) \tag{2.20}$$

Der Berechnungsablauf des SAC-Verfahrens ist in Pseudocode 2 übersichtlich dargestellt. Zunächst werden die Parameter für die neuronalen Netze der beiden Q-Funktionen $Q_{\Phi 1}$ und $Q_{\Phi 2}$ sowie der Policy π_Θ mit zufälligen Werten initialisiert (Zeile 1). Der anfangs leere Replaybuffer $\mathcal{B}$ wird erstellt (Zeile 2). Die Gewichte der Zielnetzwerke Φ'_1 und Φ'_2 werden mit den Initialwerten der anderen Netzwerke gleichgesetzt (Zeile 3). Nach der Initialisierung beginnt die eigentliche Lernphase. Der Algorithmus wählt Aktionen gemäß der Policy aus (Zeile 6). Diese Aktionen werden mit einem zufälligen Rauschen, dem sogenannten Exploration-Noise überlagert. Dadurch wird die Exploration von Bereichen, welche die Policy nicht ansteuert, gefördert. Die Tupel aus (s,a,r,s') werden in den Replaybuffer gespeichert (Zeile 7). Immer wenn ein Update der Q-Funktion ansteht, werden aus dem Replaybuffer zufällige Tupel ausgewählt und damit die Q-Netzwerke gemäß Gleichung 2.16 neu berechnet (Zeile 14). Für das Update der Policy (Zeile 17) wird der oben beschriebene Reparametrisierungs-Trick angewendet. $\tilde{a}_\Theta(s)$ ist hier die ausgewählte Stichprobe aus $\pi_\Theta(s)$. Abschließend werden die Gewichte der Zielnetzwerke neu bestimmt (Zeile 20). Dieses Schema wird bis zur Konvergenz durchgeführt.

Algorithmus 2 Soft Actor-Critic (SAC) nach [35, 39]

1: Initialisiere die Gewichte des Policy Netzes Θ und der beiden Q-Funktionen Φ_1, Φ_2
2: Initialisiere den leeren Replaybuffer $\mathcal{D}$
3: Setze die Zielparameter mit den aktuellen Parametern gleich:$\Phi'_1 \leftarrow \Phi_1, \Phi'_2 \leftarrow \Phi_2$
4: **for** $t = 1 \rightarrow T$ **do** ▷ Berechnung einer Episode
5: Wähle Aktion mit Exploration-Noise ϵ: $a = \pi_\Theta(s) + \epsilon$.
6: Führe Aktion a aus und beobachte den nächsten Zustand s' und Reward r
7: Speichere die Transition (s,a,r,s') im Replay-Buffer $\mathcal{D}$
8: **if** Zeit für Update **then**
9: Wähle zufällig einen Stapel Transitionen $\mathcal{B} = (s,a,r,s')$ aus $\mathcal{D}$
10: Berechne die Zielaktionen:
11: $\bar{a} = \pi'_\Theta(s') + \epsilon$
12: Berechne die Zielfunktion:
13: $y = r + \gamma \underset{i=1,2}{min} Q_{\Phi'_i}(s',\bar{a'}) - \alpha log \pi_\Theta(\bar{a'}|s')$
14: Update der Q-Funktion gemäß Gleichung 2.16:
15: $\nabla_{\Phi_i} = \frac{1}{|\mathcal{B}|} \sum_{(s,a,r,s') \in \mathcal{B}} \left[\left(Q_{\Phi_i}(s,a) - y \right)^2 \right]$
16: **if** Zeit für Policy-Update **then**
17: Berechne ein Update der Policy mit dem Gradient Ascent
18: $\nabla_\theta = \frac{1}{|\mathcal{B}|} \sum_{s \in \mathcal{B}} (\underset{i=1,2}{min} Q_{\Phi_i}(s,\tilde{a}_\Theta(s)) - \alpha log \pi_\Theta(\tilde{a}_\Theta(s)|s))$
19: Berechne Update der Zielnetzwerke:
20: $\Phi_i \leftarrow \rho \Phi_i + (1 - \rho) \Phi'_i$

2.4. Domain Randomization

Wie bereits in Kapitel 2.2.1 gezeigt, besteht eine Herausforderung des Reinforcement Learnings darin, das neuronale Netz aus der Simulation auf die reale Situation zu übertragen. Es ist praktisch nicht umsetzbar, einen RL-Regler direkt am Prüfstand zu trainieren. Bei der Übertragung von der Ausgangsdomäne (Simulation) in die Zieldomäne (Realität) werden zwei unterschiedliche Vorgehensweisen unterschieden.

Domain Adaptation Methoden (DA) benötigen zusätzliche Trainingsdaten in der Zieldomäne, damit die bereits trainierte Policy an die Realität angepasst werden kann. Hierbei wird vorausgesetzt, dass sich sowohl die Ausgangsdomäne als auch die Zieldomäne Charakteristiken teilen. Nur so kann ein in der Simulation erlerntes Verhalten auch in der Realität sinnvoll sein. Die resultierende, in der Realität angewendete Policy unterscheidet sich somit von der in der Simulation erlernten.

Im Unterschied dazu verwendet die **Domain Randomization** (DR) *Zero Shot Transfer Methoden*. Das bedeutet, dass die in der Simulation erlernte Policy direkt auf das Problem in der Realität angewendet wird. Hierfür werden während des Trainings entsprechende Parameter in einem festgelegten Spektrum variiert um etwaige Abweichungen in der Realität abzudecken. Für den Fall eines Raketenantriebssystems kommen als zu variierende Parameter beispielsweise der in das System geleitete Vordruck oder die Rohrlänge einzelner Leitungen in Frage. Auf diese Weise wird eine robustere Policy erlernt und Überanpassungen vermieden [25].

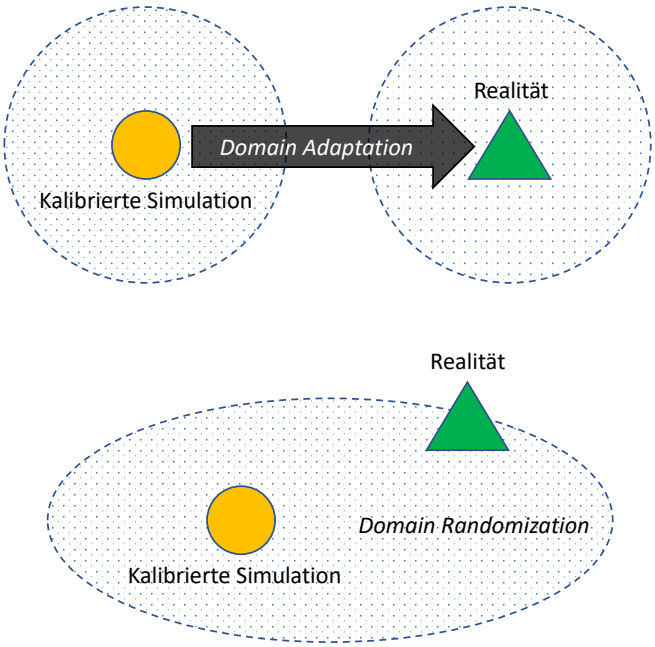


Abbildung 2.5.: Transfermethoden für einen RL-Agenten von der Simulation in die Realität nach [40].

Der Transfer zwischen Simulation und Realität der beiden Methoden Domain Adaptation und Domain Randomization ist in Darstellung 2.5 grafisch veranschaulicht. Der transparente Bereich stellt hier die Domäne dar, welche sich für die Domain Adaptation Methode in Quell- und Zieldomäne unterteilt.

In dieser Masterarbeit kommt die Domain Randomization Methode zum Einsatz. Das Training der neuronalen Netze findet somit in jeder Episode an einer etwas anderen Simulationsumgebung statt. Es wird ein Parameter μ eingeführt, der die variablen Größen enthält [5]. Die Zustandsübergangsfunktion P hängt nicht mehr nur von State s_t und Action a_t ab, sondern auch von den Parametern, die zufällig variiert werden. Dabei ist zu beachten, dass die Bandbreite der zufälligen Variationen so ausgewählt werden müssen, dass sie die erwarteten Abweichungen zwischen Simulation und Realität abdecken. Die Auswahl der variierten Parameter hat signifikanten Einfluss auf die erzielbaren Ergebnisse [41]. Außerdem sollten sie so gewählt werden, dass immer physikalisch mögliche Lösungen existieren und die Bandbreite möglichst gering gehalten wird. Bei Vernachlässigung dieser Grundsätze kann das Erlernen sinnvoller Policies erschwert werden [42].

Die Schwierigkeit bei diesen Herangehensweisen ist die Suche geeigneter Parameter und die Definition der Variationsbandbreite. Da dies ein zeitaufwendiger Prozess ist, der eine detaillierte System-Kenntniss erfordert, existieren mehrere Ansätze zur Automatisierung [5]:

Active Domain Randomization Für die Parameterauswahlstrategie wird ein Algorithmus verwendet, welcher die Trainingsumgebung nach besonders schlecht gelösten Parameterkonfigurationen durchsucht. Es wird davon ausgegangen, dass diese Konfigurationen besonders viele Informationen zur Verallgemeinerung des Problems enthalten. In diesen Bereichen wird das Training intensiviert [43].

Automatic Domain Randomization Das Training findet zunächst an einer einzelnen Simulationsumgebung mit festgelegten Parametern statt. Sobald gute Ergebnisse erzielt werden, wird nach und nach die Bandbreite der zufällig variierten Parameter erweitert. Dieser Vorgang wird so lange wiederholt, bis keine guten Ergebnisse mehr geliefert werden. Im Vergleich zu der Vorgehensweise, direkt alle Parameter im vollen Spektrum zu variieren, erhoffen sich die Autoren mit dieser Methode eine schnellere Konvergenz des Verfahrens. [9].

Structured Domain Randomization Bei dieser Vorgehensweise ist die grundlegende Idee, dass nicht jede physikalisch mögliche Variation der Simulationsumgebung sinnvoll ist. Hauptsächlich kommt diese Methode für die Generierung von visuellen Trainingsdaten zum Einsatz. Sollen beispielsweise Daten für ein selbstfahrendes Auto erstellt werden, so sind andere Autos nicht völlig zufällig in dem beobachteten Raum verteilt, sondern befinden sich auf entsprechenden Fahrspuren [44].

Anpassung mit realen Erfahrungswerten Um die Verhaltensweise der Policy an die Realität anzupassen, wird die Verteilung der Simulationsparameter mit Datensätzen aus der realen Anwendung modifiziert [42].

Es existieren bereits verschiedene Anwendungsfälle erfolgreicher RL-Agenten, welche mit Hilfe der Domain Randomization beeindruckende Ergebnisse erzielen konnten. Dazu zählen unter anderem der autonome Drohnenflug durch ein Gebäude [45] und diverse Robotermanipulationsaufgaben [9]. Auch die Regelung eines Raketenantriebssystems ist bereits in einer Simulationsumgebung gelungen und soll nun in einer realen Umgebung erprobt werden [5]. Aus diesem Anlass wird in dieser Arbeit untersucht, welchen Einfluss unterschiedliche DR-Konfigurationen auf das Training haben und ob die Regelung eines Raketenantriebssystems damit auch in der Realität verbessert werden kann.

3. Aufbau des Prüfstandmodells

Das Ziel der Arbeit ist der Aufbau und die Erprobung eines RL basierten Reglers in einer Simulationsumgebung und an einem Prüfstand für Orbitalantriebe. Für das Reinforcement Learning wird der SAC-Algorithmus verwendet und verschiedene Trainingsparameter untersucht. Diese beinhalten eine Variation des Beobachtungsraums, der Belohnungsfunktion und die Anwendung von Domain Randomization. Im Vergleich dazu wird der Einfluss von Rauschen (eng. *Noise*) verschiedener physikalischer Größen auf die Regelgüte bewertet. Dazu zählen der auf das System gegebene Vordruck, die Rohrlänge und die Ventil-Öffnungsgeschwindigkeit. Außerdem wird überprüft, welchen Einfluss eine Verzögerung (eng. *Delay*) beim Öffnen beziehungsweise Schließen des Regelventils auf die Regelung hat. Die durchgeführten Simulationen finden in dem Programm EcosimPro der Firma Empresarios Agrupados statt [46]. Diese lizensierte Software ist ein 0/1-D Simulationsprogramm zur Lösung von differential-algebraischen Gleichungen mit diskreten Events und eignet sich damit zur Systemsimulation. Für die Simulation von Raketenantriebssystemen in EcosimPro wird das Toolkit ESPSS (*European Space Propulsion System Simulation*) der Europäischen Weltraum Agentur ESA verwendet, da es bereits die geeigneten Bibliotheken für diese Art von Systemen beinhaltet. Dazu zählen mathematische Modelle für verschiedene Komponenten eines Raketenantriebssystems, wie zum Beispiel Brennkammern, Rohrleitungen oder Ventile. Mit Hilfe dieses Simulationsprogramms wird ein Modell des Prüfstandsaufbaus am M11.5 des Deutschen Zentrums für Luft- und Raumfahrt in Lampoldshausen aufgebaut und im Anschluss mittels Reinforcement Learning in Python geregelt. Der Aufbau des realen Prüfstands am DLR sowie der Aufbau des Simulationsmodells und die Umsetzung des Reinforcement Learnings in Python sind in den folgenden Abschnitte beschrieben. Abschließend wird in diesem Kapitel die Validierung des Simulationsmodells unter Verwendung von realen Prüfstandsdaten vorgestellt.

3.1. Prüfstandsaufbau am DLR

Das Simulationsmodell soll die Versuchsbrennkammer und deren Versorgungssystem für einen vereinfachten, experimentellen Kaltgas-Versuch mit Stickstoff abbilden. Bei der Brennkammer handelt es sich um eine modular gestaltete, regenerativ gekühlte, 3D-gedruckte Brennkammer aus einer Kupferlegierung (CuCrZr), welche für die Treibstoffe Lachgas und Ethan (HyNOx) entwickelt wurde [47]. Aufgrund des modularen Aufbaus ist es möglich, verschiedene Injektoren und Brennkammerlängen zu testen. Außerdem ist es möglich, die Brennkammer sowohl im Monopropellant als auch im Bipropellant-Modus zu betreiben. Es kann somit entweder vorgemischter Treibstoff direkt eingespritzt werden

oder beide Stoffe einzeln, wobei die Vermischung dann erst im Brennraum stattfindet [5]. Das Simulationsmodell und der Prüfstandsversuch beschränken sich in dieser Arbeit auf den Betrieb mit gasförmigem Stickstoff, welcher unter normalen Umständen zur Spülung des Systems verwendet wird. Zur Überprüfung des RL-basierten Regelverfahrens genügt diese Konfiguration. Die Brennkammer ist nach einem solchen Kaltgasversuch fotografiert worden 3.1. In der Fortführung dieser Arbeit sollen die RL basierten Regler auch in Heißgastests mit HyNOx als Treibstoff eingesetzt werden.

Abbildung 3.1.: Versuchsbrennkammer nach einem Kaltgasversuch mit Stickstoff

Die 3D-gedruckte Brennkammer ist [48] nachempfunden und besitzt vier Kühlkanäle mit einem kreisförmigen Querschnitt von je 3 mm. Der Brennkammerdurchmesser beträgt 14 mm, der Düsenhalsdurchmesser 4,2 mm. Die Temperatur des Kühlmediums kann am Auslass der Kühlkanäle mittels zweier Thermoelemente (Typ K) bestimmt werden. Zusätzlich dienen zwölf Thermoelemente zur Messung der Brennkammerwandtemperatur an verschieden radialen und axialen Positionen. Ein Drucksensor misst während des Versuchs den statischen Druck in der Brennkammer. Der geplante Schub von 22 N wird bei einem Gesamtmassenstrom von 7,48 g/s erreicht [48].

Die Triebwerksversorgung mit Stickstoff erfolgt über Druckgasflaschen. Es werden keine Pumpen benötigt. Der Stickstoff wird am Prüfstandskomplex M11 unter einem Druck von 200 bar zentral gelagert und dem Versuchsaufbau über eine Leitung zugeführt. Das entnommene Gas durchströmt zunächst einen Druckminderer, um den Zuleitungsdruck zum Triebwerk definiert einstellen zu können. Die Massenströme ins Triebwerk werden mit Coriolis-Messgeräten des Herstellers Rheonik bestimmt. Kurz vor der Brennkammer sind im Lachgas- und Ethan-Strang jeweils Regelventile vom Typ MPG 03 PR des Herstellers m-tech angebracht, welche die Feinjustierung der Massenströme ermöglichen. In der Kaltgaskonfiguration wird nur eines dieser Leitungssysteme benutzt.

Abbildung 3.2.: Testcontainer mit Prüfstand für grüne Treibstoffe auf dem Testfeld M11.5 aus [12]

Der gesamte Prüfstandsaufbau mit Messsystemen befindet sich in einem Container, welcher in Abbildung 3.2 zu sehen ist. Der Container ist durch eine Trennwand in zwei Bereiche unterteilt. Auf der Vorderseite befindet sich der Versuchsaufbau mit Brennkammer, auf der Rückseite das Mess- und Steuersystem. Das Messsystem zeichnet während eines Versuchs die verfügbaren Sensordaten auf und sendet Stellsignale an die Ventile und den Druckminderer. Die analogen Druck- und Temperaturdaten werden mit einem Dewetron Messverstärker erfasst, an ein Adwin-System weitergeleitet. Im Adwin-System findet die Messdatenerfassung, Digitalisierung und Steuerung statt. Die ausgegebenen Messdaten werden an ein in Python entwickeltes Benutzerinterface weitergeleitet, in welchem die Sequenzen für die Ventilpositionen während der Versuche eingestellt werden können [49].

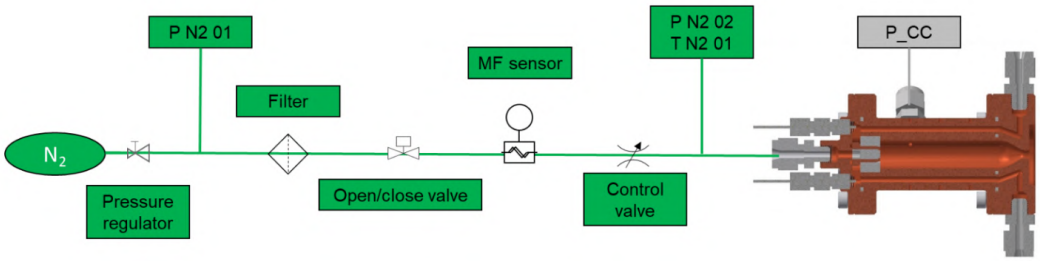


Abbildung 3.3.: Fluidplan für die Kaltgaskonfiguration am Testfeld M11.5 aus [49]

Abbildung 3.3 zeigt den Fluidplan mit allen entsprechenden Bauteilen, Messstellen und Leitungen für die Prüfstandskonfiguration mit Stickstoff. Eine detaillierte Beschreibung der Komponenten wird anhand des Simulationsmodells im folgenden Abschnitt 3.2 beschrieben.

3.2. Simulationsmodell in EcosimPro/ESPSS

Der in 3.1 beschriebene Versuchsaufbau wurde in EcosimPro/ESPSS in ein Simulationsmodell übertragen. Hierfür können die einzelnen Komponenten im grafischen Benutzerinterface (eng. *Graphical User Interface*, GUI) des Programms durch Verbinden miteinander in physikalische Beziehung gesetzt werden. Der schematische Aufbau der Prüfstandskonfiguration in der Simulationsumgebung ist in Abbildung 3.4 zu sehen.

Am Prüfstand beginnt der Aufbau mit einem Tank, gefolgt von einem Tankventil und einer Verbindungsleitung zwischen Tank und Druckregler. Dieser Teil wird in der Simulation durch eine Druckrandbedingung modelliert, welche einen Vordruck auf Leitung (1) gibt. Die Ziffern (2), (4) und (6) kennzeichnen die weiteren Verbindungsrohre. Das Schussventil (3) kann entweder vollständig geöffnet oder vollständig geschlossen werden und sorgt für die Bedruckung der dahinterliegenden Komponenten. Das mit RL zu regelnde Regelventil (4) kann jeden Öffnungszustand einnehmen. Somit kann der Druck in der Brennkammer (7) reguliert werden. Alle Komponenten sind geometrisch an die real verbauten Teile angepasst. Dazu zählen die Durchmesser und Längen der Rohre und die Brennkammergeometrie.

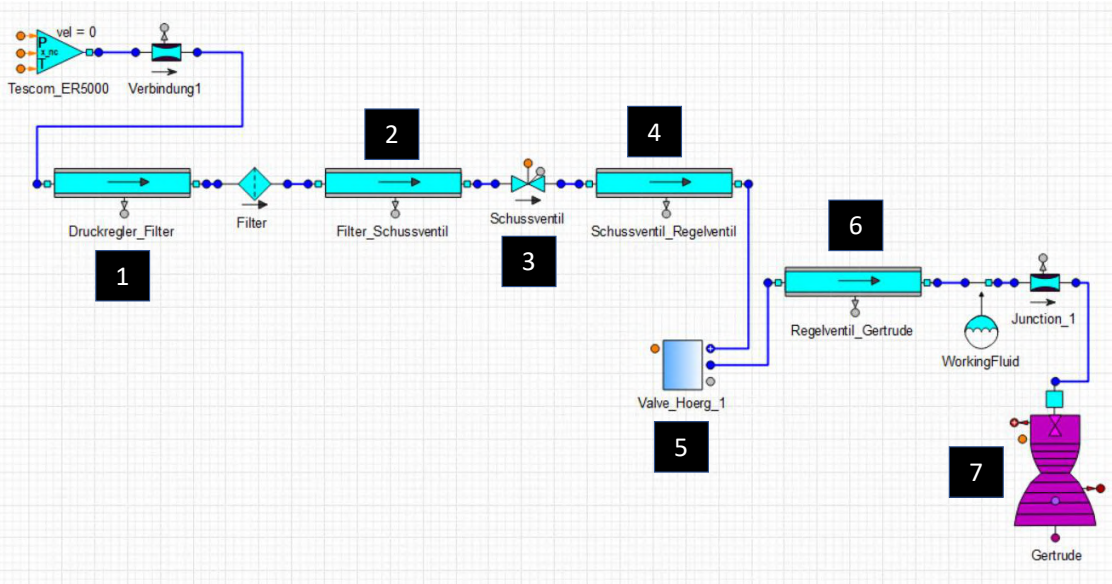


Abbildung 3.4.: GUI von EcosimPro/ESPSS, Darstellung des Kaltgas-Simulationsmodells für die Versorgung der Versuchsbrennkammer mit Stickstoff

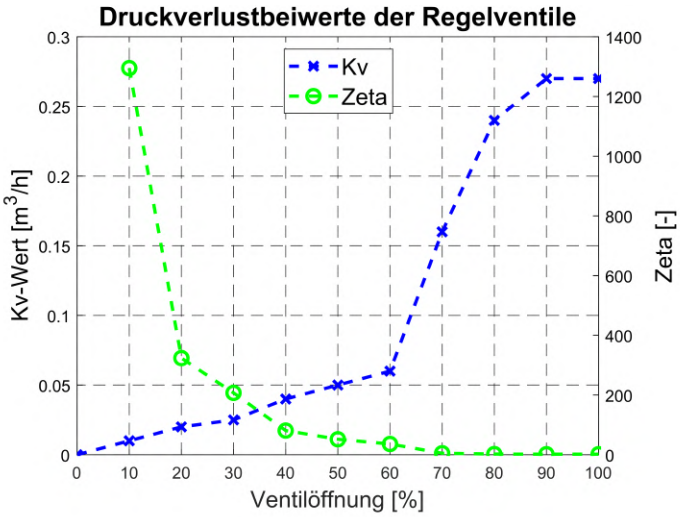
Der Druckverlust des Regelventils bei unterschiedlichen Ventilstellungen und der zeitliche Verlauf des Öffnungsvorgangs müssen möglichst exakt modelliert werden, um die Dynamik des Simulationsmodells an die des realen Prüfstands anzupassen. Das Schussventil wird ausschließlich bei der Zündung und beim Abschalten des Triebwerks genutzt und ist während des Betriebs komplett geöffnet. In dieser Arbeit liegt der Schwerpunkt auf der Regelung während des Betriebs. Daher ist es nicht notwendig, die Dynamik des Schussventils zu modellieren [5].

Während des Betriebs des Triebwerks kann der Massenstrom und somit der Brennkammerdruck durch das Öffnen des Regelventils gesteuert werden. Um den Druckverlust durch das Ventil zu modellieren wird die Widerstandsziffer ζ benötigt. Da im Datenblatt [50] des verwendeten Regelventils nur der ventilhubabhängige k_v-Wert hinterlegt ist, muss der k_v-Wert mit Hilfe des hydraulischen Durchmessers d_h nach Gleichung 3.21 in die Widerstandsziffer umgerechnet werden [51]. Der k_v Wert gibt die Durchflussrate von Wasser in m^3/h bei einer Temperatur zwischen 5 und 20 °C an, der bei einem Druckabfall von 1 bar durch das Ventil strömt [52].

$$\zeta = 1{,}5988 * 10^{-3} * \frac{d_h^4}{k_v^2} \tag{3.21}$$

Der beschriebene Berechnungsvorgang und die Modellierung des Regelventils in der Simulationsumgebung EcosimPro/ESPSS wurde bereits im Vorfeld der Arbeit durchgeführt und wird im Folgenden erläutert [5].

Pos. [%]	k_v-Wert [m^3/s]	ζ [-]
0	0	–
10	0,01	1295,03
20	0,02	323,76
30	0,03	207,20
40	0,04	80,94
50	0,05	51,80
60	0,06	35,97
70	0,16	5,06
80	0,24	2,25
90	0,27	1,78
100	0,27	1,78



Tab. 3.1.: Druckverlustkennlinien der im Prüfstand verbauten Regelventile aus [5]

In Tabelle 3.1 sind die aus dem Datenblatt [50] abgelesenen k_v-Werte sowie die daraus berechneten Widerstandsbeiwerte ζ tabelliert. Die Widerstandsbeiwerte sind anschließend in EcosimPro als Tabelle in Abhängigkeit von der Ventilöffnung hinterlegt worden.

Dadurch wurde erreicht, dass das simulierte Ventil in jeder Position denselben Druckverlust besitzt wie das reale Ventil.

Neben der Modellierung des Druckverlustes muss der Öffnungs- und Schließvorgang des Ventils in Abhängigkeit von der Zeit beschrieben werden. In EcosimPro ist diese Funktion durch eine Übertragungsfunktion erster Ordnung implementiert, für welche die Zeitkonstante entsprechend angepasst werden muss. In [5] konnte anhand der gemessenen Sprungantwort des Regelventils für unterschiedliche Öffnungsszenarien gezeigt werden, dass sich die in EcosimPro hinterlegte Übertragungsfunktion nicht dafür eignet, das zeitliche Verhalten der real am Prüfstand verbauten Regelventile darzustellen. Daher ist im Programm stattdessen eine angepasste Instanz der Ventilkomponente erzeugt worden, welche eine lineare Öffnungskennlinie mit einer Zeitkonstante von 0,28 besitzt [5].

3.3. RL-Programmstruktur in Python

Für das Reinforcement Learning wird in dieser Arbeit der in Kapitel 2.3.2 vorgestellte SAC-Algorithmus verwendet. Dieser ist bereits in dem Open Source Framework *Ray* enthalten und muss somit nicht selbst implementiert werden [53]. Ray liefert Bibliotheken für Maschine-Learning-Anwendungen im Allgemeinen. Für das Reinforcement Learning bietet die Bibliothek *Rllib* [54] eine große Auswahl an RL-Algorithmen. Von Vorteil für rechenintensive Prozesse wie das Training eines neuronalen Netzes ist die Möglichkeit der Parallelisierung in Ray. Das bedeutet, dass sowohl auf mehreren Kernen einer einzelnen Maschine als auch auf einem Cluster gerechnet werden kann. Somit können mehrere Prozesse zur Sammlung von Trainingsdaten simultan laufen. Ein solcher Prozess wird als *Worker* bezeichnet. Da EcosimPro nicht Multicore-fähig ist, besteht ein Worker physikalisch gesehen jeweils aus einem Rechenkern. Pro Worker wird das EcosimPro Modell einmal geladen und simuliert. Wird ein Lernprozess für ein neuronales Netz beispielsweise mit vier Workern gestartet, so werden vier Instanzen des EcosimPro Modells initialisiert und simuliert. Auf diese Weise können in der gleichen Zeit deutlich mehr Trainingsdaten erzeugt werden, als wenn die Daten aus einem einzelnen Prozess generiert werden.

Der Reinforcement Learning Algorithmus kann über eine in Python programmierte Schnittstelle auf das EcosimPro Simulationsmodell zugreifen. Die Implementierung der Schnittstelle, der sogenannten *rocket control library*, erfolgte bereits DLR-intern. Um diese anzuwenden, muss das in EcosimPro/ESPSS aufgebaute Experiment zunächst in ein sogenanntes *Deck* überführt werden. Ein Deck ist eine eigenständige, ausführbare Datei, die unabhängig vom Programm genutzt werden kann. Es beinhaltet als Eingabewerte die jeweils neuen Regelventilpositionen. Gemäß den zugrundeliegenden Gleichungen werden in jedem Zeitschritt die Ergebnisse berechnet und als Ausgabewerte zurückgegeben. Dadurch stellt sich nach jedem Zeitschritt heraus, wie sich die Betriebsgrößen des Triebwerks ändern [5]. Im Rahmen dieser Arbeit wird als ausgewählte Größe der Brennkammerdruck zurückgegeben, da dieser auf einen stationären Wert geregelt werden soll.

Der beschriebene Aufbau eignet sich für eine Reinforcement Learning Umgebung, da die Ein- und Ausgabegrößen als Action- und Observation-Space definiert werden

können. Das Prinzip des Datenaustauschs zwischen dem Simulationsprogramm und dem RL-Agenten in Python ist in Schema 3.5 dargestellt.

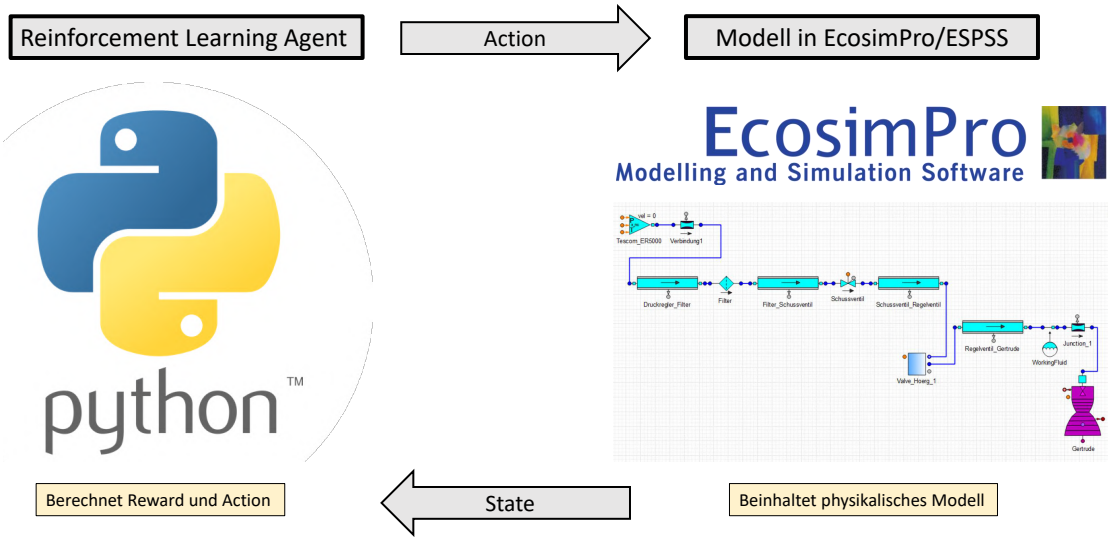


Abbildung 3.5.: Datenaustausch zwischen EcosimPro (Simulation) und Python (RL-Algorithmus) nach [5]

Der Reinforcement Learning Agent ist in Python implementiert und berechnet den Reward und die Action anhand entsprechend programmierter Funktionen. Die Action dient als Eingangsgröße des EcosimPro Modells. Dort wird anhand der zugrundeliegenden physikalischen Gleichungen der Zustand des Systems für den nächsten Zeitschritt berechnet. Ein Teil der Zustandsdaten wird als State an den Agenten geleitet [5]. In dieser Arbeit wird eine Zeitschrittgröße von 0,1 s und eine Episodenlänge von 100 Zeitschritten definiert. Somit entspricht eine Episode einer Laufzeit von 10 s am Prüfstand.

Für die Einbindung des Decks in das RL-Framework in Python wird es in eine abstrakte Umgebung überführt. Hierfür eignet sich *Gym* der Firma OpenAI [55]. Gym ist eine Standard-API (*Application Programming Interface*) für RL-Prozesse und bietet eine große Sammlung von Referenzumgebungen. Sie enthält standardisierte Funktionen, welche die Kommunikation zwischen Algorithmus und Simulationsmodell ermöglichen. Diese Funktionen werden nacheinander aufgerufen und erzeugen so eine Iterationsschleife des RL-Algorithmus. Der Anwender kann festlegen, aus wie vielen Zeitschritten eine Iterationsschleife bestehen soll. Nach einer bestimmten Anzahl von Zeitschritten endet der Prozess. Der Ablauf einer solchen Iterationsschleife mit den entsprechenden Funktionsaufrufen in Python wird in Abbildung 3.6 schematisiert. Im Folgenden werden zunächst die verwendeten Funktionen erläutert [5]:

set controls Setzt den Sollwert der Regelventilposition im Simulationsmodell auf den Wert, welcher vom RL-Algorithmus als Action bestimmt wurde.

step Simuliert einen Zeitschritt von 0,1 s im EcosimPro-Deck.

get obs Liest die Ergebnisse des letzten Simulationsschrittes aus und übernimmt diese in den Observation-Space.

get reward Berechnet den Reward zum jeweiligen Zeitschritt.

render Erzeugt Diagramme zur Visualisierung des Trainingsfortschritts.

done Überprüft, ob eine Episode beendet ist und neu gestartet werden kann. Die Episodenlänge wird auf 100 Zeitschritte entsprechend 10 s Triebwerksbetrieb definiert.

reset Setzt das Simulationsmodell am Ende einer Episode auf den Anfangszustand zurück, damit eine neue Episode gestartet werden kann.

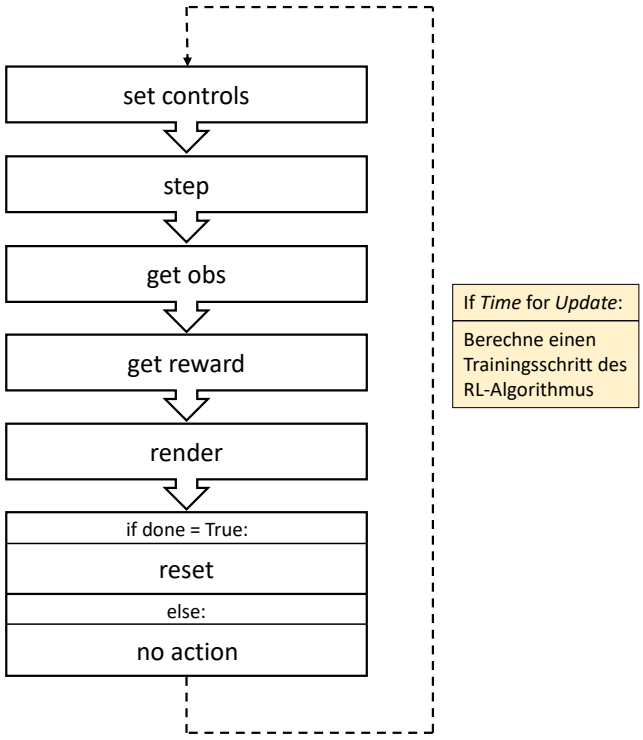


Abbildung 3.6.: Berechnungsschema einer Iterationsschleife des RL-Algorithmus in Python

Ziel der Simulation ist es, den Brennkammerdruck der Versuchsbrennkammer auf einen statischen Wert regeln zu können. Hierfür müssen zunächst **Observation-Space, Action-Space** und **Reward** definiert werden. Dies geschieht in einem ersten Schritt für ein in 4.1 ausgewertetes neuronales Netz, welches als Referenz für die Auswertung der anderen untersuchten neuronalen Netze verwendet wird. Dieses neuronale Netz wird im folgenden Verlauf dieser Arbeit als Standardmodell bezeichnet.

Folgende Größen dienen dem Standardmodell als **Observation-Space**:

P_{BK}	Brennkammerdruck
P_{DR}	Druck im Rohr stromab des Druckreglers
P_{RV}	Druck im Rohr stromab des Regelventils
T_{RV}	Temperatur im Rohr stromab des Regelventils
pos	Ist-Wert des Regelventils
pos'	Ist-Wert der Regelventil-Öffnungsgeschwindigkeit

Im Rahmen der durchgeführten Parameterstudien wird der Beobachtungsraum genauer untersucht um festzustellen, welche der enthaltenen Größen zielführend sind und welche nur einen geringen oder gar keinen Einfluss auf das Ergebnis haben. Dafür werden neuronale Netze mit unterschiedlichen Beobachtungsräumen trainiert und die erzielten Ergebnisse in Abschnitt 4.1.2 miteinander verglichen.

Der **Action-Space** besteht in der Simulation aus einer Funktion, welche von der Regelventilposition pos abhängt. Der Agent hat nur auf diese Größe Einfluss. Alle übrigen Ventile des Systems sind fixiert und im Betrieb komplett geöffnet. Das Regelventil kann prinzipiell zwischen 0 % und 100 % geöffnet sein. Um eine Erkundung des RL-Agenten in Bereichen, die nicht zielführend sind, zu vermeiden, wird der Handlungsspielraum auf Ventilstellungen zwischen 10 % und 90 % limitiert. Erfahrungswerte haben gezeigt, dass die Ventilposition für den gewollten Brennkammerdruck in diesem Bereich liegt. Um den Regelprozess des Brennkammerdrucks zu stabilisieren, wird die Action-Funktion so implementiert, dass das Ventil pro Zeitschritt um maximal 5 % aufgefahren bzw. geschlossen werden kann. Der Anlass für diese Vorgehensweise wird aus den Ergebnissen des Standardmodells in Abschnitt 4.1 ersichtlich.

Der **Reward** des Standardmodells setzt sich aus zwei Anteilen zusammen und kann aus der folgenden Gleichung berechnet werden:

$$rew = rew_1 + rew_2 \tag{3.22}$$

Der erste Teil (Gleichung 3.23) der Rewardfunktion beinhaltet die Abweichung des Brennkammerdrucks P_{BK} von dessen vorgegebenem Sollwert P_{BKsoll}. Es wird eine Wurzelfunktion verwendet, da sie im Bereich des Sollwertes einen größeren Gradienten liefert als eine lineare Funktion [5]. Entspricht der Brennkammerdruck genau dem Sollwert, wird der maximale Wert $rew_1 = 0$ erreicht. Je höher die Abweichung vom Sollwert, desto kleiner wird der rew_1-Anteil.

$$rew_1 = -\sqrt{|P_{BK} - P_{BKsoll}|} \tag{3.23}$$

Der zweite Summand rew_2 der Rewardfunktion soll das Verhalten des Agenten zusätzlich belohnen, sobald dieser in einem Zeitschritt das Regelziel erfüllt. Sollte die Abweichung des Brennkammerdrucks P_{BK} kleiner als 0,1 bar vom Sollwert P_{BKsoll} sein, wird der

Agent als erfolgreich angesehen. Die Validierung des Simulationsmodells in 3.4 zeigt, dass die gewählte Abweichung von 0,1 bar innerhalb der Simulationsungenauigkeit liegt, der zwischen Simulationsmodell und realem Prüfstand gemacht wird. Daher wird rew_2 so definiert, dass eine Bonusbelohnung vergeben wird, sobald die Abweichung der beiden Drücke P_{BK} und P_{BKsoll} für drei Zeitschritte hintereinander geringer als 0,1 bar ist:

$$\textbf{if} \quad |P_{BK} - P_{BKsoll}| < 0{,}1\,\text{bar} \tag{3.24}$$

$$\textbf{and} \quad numgoodtimesteps > 3 \tag{3.25}$$

$$rew_2 = 100$$

$$\textbf{else}$$

$$rew_2 = 0$$

Auf diese Weise kann gewährleistet werden, dass der Agent beim Erreichen eines gewollten Bereichs stabilisiert wird. Außerdem lässt sich anhand dieser Definition des Rewards für den Anwender ein leichterer Bezug zwischen Zahlenwert und Lösungsgüte herstellen. Anhand der Größenordnung des Rewards einer Episode kann einfach abgeschätzt werden, ob und nach wie vielen Zeitschritten das Regelziel erfüllt ist. Wird für eine Episode aus 100 Zeitschritten beispielsweise ein Gesamt-Reward (eng. *cumulative reward*) von 10.000 erreicht, würde dies bedeuten, dass in jedem Zeitschritt das Regelziel erfüllt wurde und keine Abweichung zwischen P_{BK} und P_{BKsoll} existiert. Dies ist der theoretische Maximalwert der Rewardfunktion. Ein negativer Wert weist darauf hin, dass es die Reglung in keinem Zeitschritt erfolgreich geschafft hat, die Schwelle von 0,1 bar Abweichung zu unterschreiten.

3.4. Validierung der Simulation

Da die in dieser Masterarbeit trainierten neuronalen Netze nicht nur in der Simulation, sondern auch am realen Prüfstand getestet werden sollen, muss die Abweichung zwischen Simulationsmodell und Versuchsaufbau möglichst klein sein. Um das Simulationsoffset zwischen dem in EcosimPro erstellten Modell und den am Prüfstand gemessenen Testdaten zu überprüfen, wird eine Validierung des Brennkammerdrucks P_{BK}, des Stickstoff-Massenstroms $\dot{m}$ und der Drücke in den Rohren hinter dem Druckregler und dem Regelventil durchgeführt.

Hierfür werden in der Simulation die identischen Einstellungen wie im Versuch am Prüfstand gewählt. Das Regelventil wird auf die gleiche Position *pos* gestellt und der Sollwert des Druckreglers P_{soll} wird auf den selben Wert gesetzt wie im entsprechenden Vergleichsversuch. Dabei wird in allen Tests ein Vordruck von 80 bar verwendet. Um ein breites Band aller gängigen Betriebspunkte des Triebwerks zu überprüfen, werden verschiedene Brennkammerdrücke zwischen 3 bar und 10 bar in diskreten Schritten von 0,5 bar angefahren. Hierfür wird in der Simulation die Regelventilposition so vorgegeben, dass sich der jeweils gewünschte Brennkammerdruck ergibt. Anschließend wird gemessen, welcher Brennkammerdruck mit der identischen Ventilposition am Prüfstand

herrscht. Tabelle 3.2 zeigt eine Übersicht der Ergebnisse aller untersuchten Größen für drei exemplarische Ventilöffnungen.

Bezeichnung	pos [%]	P_{BK} [bar]	P_{DR} [bar]	P_{RV} [bar]	$\dot{m}_N$ [g/s]
Val Sim 1	7,5	3,04	79,17	11,51	10,0
Val V 1	7,5	2,89	78,80	10,80	9,6
Val Sim 2	39	6,03	76,71	22,82	20,0
Val V 2	39	5,75	74,50	21,00	19,8
Val Sim 3	89	10,01	70,77	37,86	33,0
Val V 3	89	9,30	64,20	34,00	32,3

TAB. 3.2.: Vergleich der Simulationsdaten (grau hinterlegt) mit den Messdaten am Prüfstand

Der Vergleich zwischen Simulation und Versuch zeigt, dass der Simulationsfehler der gemessenen Drücke im Fall der maximalen untersuchten Ventilöffnung von 89 % in Versuch Val V 3 am höchsten ausfällt. Die prozentuale Abweichung des Brennkammerdrucks beträgt in diesem Fall 7 %, was einer Druckdifferenz von 0,7 bar entspricht. Dies ist die höchste absolute Abweichung der gesamten Versuchsreihe für den Brennkammerdruck. Die Tendenz des steigenden Simulationsfehlers beim Öffnen des Ventils lässt sich für die drei untersuchten Drücke P_{BK}, P_{DR} und P_{RV} in den entsprechenden Schaubildern 3.7, A1 und A2 erkennen.

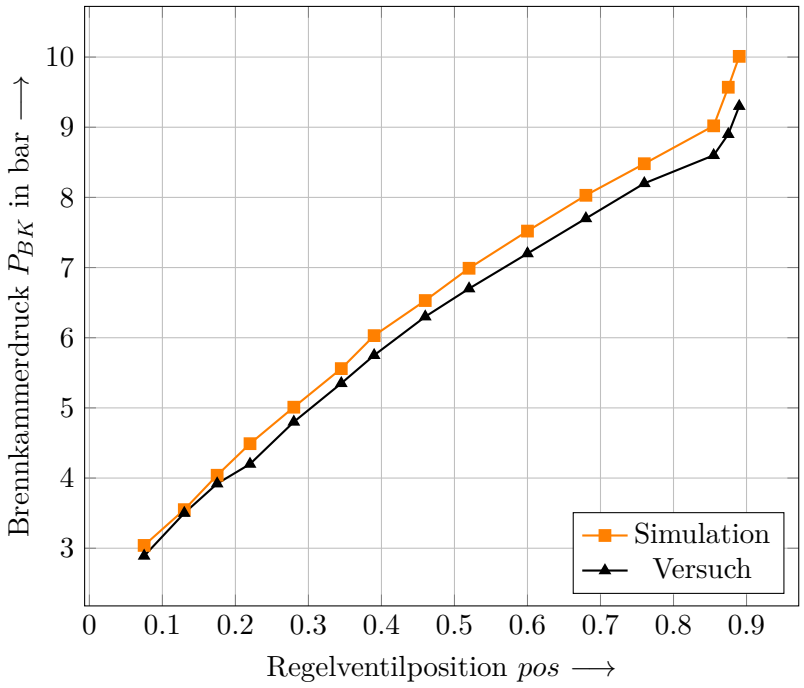


ABBILDUNG 3.7.: Vergleich des Brennkammerdrucks von Simulation und Prüfstandsversuch

Der RL-Regler soll in dieser Arbeit einen stationären Brennkammerdruck einstellen können. Hierfür wurde als Zielgröße ein Druck von 6 bar gewählt, welcher in etwa der Mitte des üblichen Betriebsbereichs der Brennkammer liegt. Bei dem gewählten Brennkammerdruck beträgt die prozentuale Abweichung zwischen Simulation und Versuch 4,6 %, was einer absoluten Abweichung von 0,28 bar entspricht.

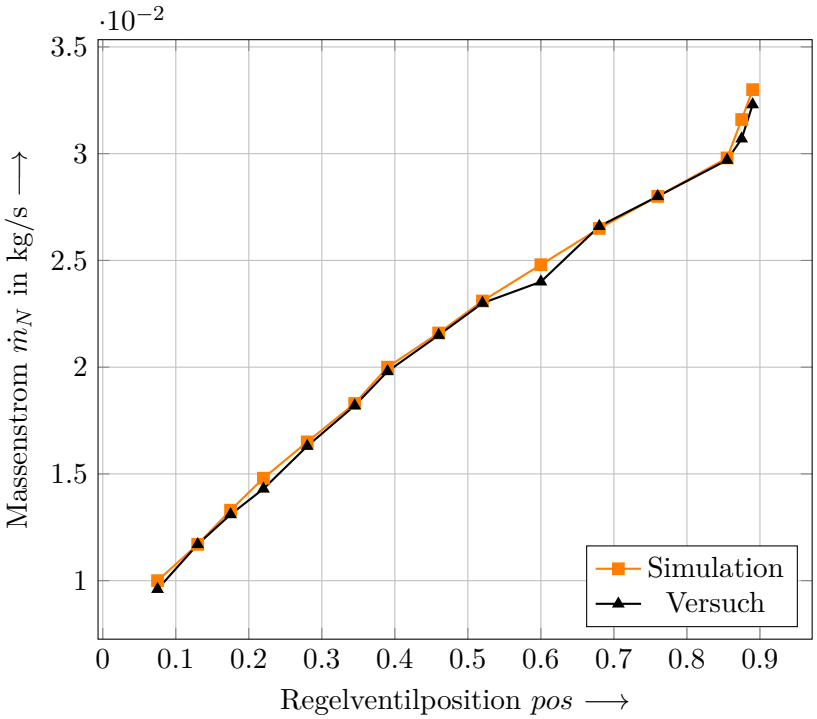


Abbildung 3.8.: Vergleich des Stickstoff-Massenstroms von Simulation und Prüfstandsversuch

Diagramm 3.8 zeigt den Vergleich des Stickstoff-Massenstroms $\dot{m}_N$ für die beschriebene Versuchsreihe. Im Gegensatz zu den Druckgrößen kann hier keine Tendenz ausgemacht werden, dass die Simulation immer zu geringe oder zu hohe Werte liefert. Mit Ausnahme bei einer Ventilöffnung von 60 % ist der prozentuale Simulationsfehler immer geringer als 3 %.

Die resultierenden Abweichungen zwischen Simulation und Realversuch können verschiedene Gründe haben. Zum einen ist die Umgebungstemperatur und die Temperatur der Rohrleitungen nicht bekannt. Die Temperatur der Komponenten nimmt im Verlauf der Kaltgastests zunehmend ab. Werden mehrere Versuche hintereinander abgehalten, sind die Bedingungen nie exakt gleich. Vor allem in Bezug zur Temperatur kommt es hier zu Schwankungen. Dies beeinflusst die Dichte des darin strömenden Gases und somit die Druckverluste. Die Modellierung der Druckverluste über den Injektor ist auch fehlerbehaftet, da es in EcosimPro kein explizites Modell für poröse Injektoren gibt, wie sie in

der Versuchsbrennkammer eingebaut sind [5]. Hier kann in der Simulationsumgebung lediglich die Injektorfläche und ein Druckverlustbeiwert angegeben werden.

Weitere Unsicherheiten ergeben sich aus den unbekannten Reibungsbeiwerten und Krümmungswinkeln der verbauten Leitungen und Schläuche, welche mit vertretbarem Aufwand in dieser Arbeit nicht bestimmt werden können. Außerdem sind die Hysterese der im Prüfstand verbauten Regelventile sowie der Messfehler der genutzten Sensorik nicht genau bekannt.

4. Ergebnisse der RL basierten Regelung

In diesem Kapitel wird untersucht, ob ein RL basierter Regler den Brennkammerdruck auf einen stationären Wert von 6 bar regeln kann. Wie bereits in Abschnitt 3.4 erläutert, stellt der gewählte Brennkammerdruck von 6 bar einen gängigen Betriebsdruck der am Prüfstand verwendeten Brennkammer dar. Das Regelziel wird als erfüllt definiert, wenn eine maximale Abweichung vom Solldruck $P_{BK,soll}$ von 0,1 bar unterschritten und im weiteren Betriebsverlauf eingehalten werden kann. Der gewählte Fehler von 0,1 bar liegt mit einer prozentualen Abweichung von rund 1,7 % innerhalb des in Kapitel 3.4 bestimmten Simulationsfehlers von 4,6 %.

Für die Regelung kommen 42 im Rahmen dieser Arbeit trainierte neuronale Netze zum Einsatz, welche zunächst in der Simulation getestet werden. Dabei werden Studien zu den folgenden Bereichen vorgestellt:

obs	Variation des Observation-Space
rew	Test verschiedener Reward-Funktionen
rand	Domain Randomization mit diversen Parametern
noise	Einbringung verschiedener Störgrößen in das Training
delay	Untersuchung von Ventilverzögerungen

Die in fett gedruckte Kurzbezeichnung dient im weiteren Verlauf zur Benennung der vorgestellten neuronalen Netze.

Alle vorgestellten Simulationen und Tests wurden mit der SAC-Implementierung innerhalb des Ray-Frameworks [53] durchgeführt. Die verwendeten Parameter zur Konfiguration des Verfahrens sind in A1 beschrieben. Da das Training eines RL-Agenten eine große Anzahl an Trainingsdaten benötigt [4], wird jedes neuronale Netz mit einer Einstellung von 200.000 Zeitschritten trainiert. Die Rechenzeit eines Zeitschrittes in der EcosimPro Simulation beträgt etwa 1 s. Damit ergibt sich eine Rechenzeit von mehr als zwei Tagen, falls die Berechnung auf einem Rechenkern ausgeführt wird. Um die Rechenzeit zu verkürzen wurde die Möglichkeit der Parallelisierung in Ray verwendet, wodurch auf mehreren Kernen gleichzeitig gerechnet werden kann. Die in dieser Arbeit vorgestellten Berechnungen wurden jeweils mit 4 Kernen durchgeführt, wodurch die Rechendauer auf etwa 15 Stunden reduziert werden konnte. In Zukunft sind Berechnungen mit deutlich mehr Rechenkernen denkbar. Hierzu sollten vorher jedoch weitere Untersuchungen stattfinden, inwiefern die Anzahl der Rechenkerne die Konvergenz und Probeneffizienz des Verfahrens beeinflusst. Ein erster Vergleich wird in Abschnitt 4.1 vorgestellt

Da die Auswahl der Zeitschrittgröße, innerhalb der der Agent seine Aktionen vorgibt, einen Einfluss auf die Rechenzeit des Trainingsprozesses hat, muss hier ein geeigneter

Kompromiss gefunden werden. Wird der Zeitschritt klein gewählt, gibt der Agent in kurzen Abständen einen neuen Sollwert für die Regelventilposition vor. Diese Wahl ist für die hier stattfindenden Untersuchungen sinnvoll, da es aufgrund der schnellen Änderungen des Strömungsquerschnitts im Prüfstandsmodell zu starken Schwankungen in Massenstrom und Druck des strömenden Fluids kommt. Wird der Zeitschritt zu groß gewählt, kann der Agent nicht mehr angemessen auf Änderungen im Betriebspunkt reagieren. Zudem können in diesem Fall pro Zeiteinheit weniger Erfahrungsdaten gesammelt werden. Allerdings steigt die Rechendauer pro simulierter Zeiteinheit an. Aus den bisher gesammelten Erfahrungen und Versuchen hat sich ein Zeitschritt von 0,1 s als guter Kompromiss zwischen Regelfrequenz, Rechendauer pro simulierter Zeiteinheit und Trainingsdatenpunkten pro simulierter Zeiteinheit herausgestellt [5]. Daher wird für die Berechnungen in dieser Arbeit eine Zeitschrittgröße von 0,1 s gewählt.

Im anschließenden Abschnitt wird ein Referenzmodell spezifiziert, welches als Vergleich für alle weiteren untersuchten neuronalen Netze herangezogen wird.

4.1. Training am Standardmodell

Für das Standardmodell gelten die in Abschnitt 3.3 beschriebenen Einstellungen bezüglich Observation-Space, Action-Space und Reward-Funktion. Der Beobachtungsraum ist in diesem Fall mit 6 enthaltenen Größen P_{BK}, P_{DR}, P_{RV}, T_{RV}, $valvepos$ und $valvepos'$ konservativ gewählt worden. In 4.1.2 wird untersucht, inwiefern die einzelnen Komponenten zielführend für den Regelerfolg sind. Das Standardmodell ist ohne Domain Randomization oder Störgrößen trainiert worden und berücksichtigt keine Verzögerung beim Auffahren des Regelventils.

Mit der gewählten Zeitschrittgröße von 0,1 s und einer festgelegten Episodenlänge von 100 Zeitschritten entspricht eine Episode einem Triebwerksbetrieb von 10 s. Der Trainingsprozess besteht aus 200.000 Zeitschritten, was umgerechnet 2.000 Trainingsepisoden entspricht. Mit der implementierten Rewardfunktion, welche in Abschnitt 3.3 beschrieben wurde, ist pro Zeitschritt ein maximaler Reward von 100 möglich. Da der Agent erst eine Sekunde nach der Zündung in die Triebwerksregelung eingreift, kann pro Episode ein maximal aufsummierter Reward (*Return*) von 9.000 erreicht werden. Dies würde voraussetzen, dass der angestrebte Brennkammerdruck von 6 bar umgehend innerhalb eines Zeitschritts erreicht wird.

In Abbildung 4.1 ist der Verlauf des Returns pro Episode während des Trainings des Standardmodells aufgezeichnet. Die Daten zur Evaluation des Trainingsverlaufs wurden mit dem Tool *TensorBoard* erzeugt [56]. Das Schaubild zeigt, dass der Return nach 20.000 Zeitschritten nicht weiter ansteigt. Da ein positiver Return erreicht wird, ist aufgrund der Definition der Rewardfunktion davon auszugehen, dass der Schwellwert σ für mehr als 3 Zeitschritte unterschritten wurde. Nur in diesem Fall ermöglicht die Reward-Funktion, einen positiven Aufschlag von 100 pro erfolgreichem Zeitschritt zu vergeben. Da der Return pro Episode gegen einen Wert von etwa 8.000 konvergiert, kann damit gerechnet werden, dass das Regelziel erfüllt und in ungefähr 80 Zeitschritten pro Episode eingehalten wird.

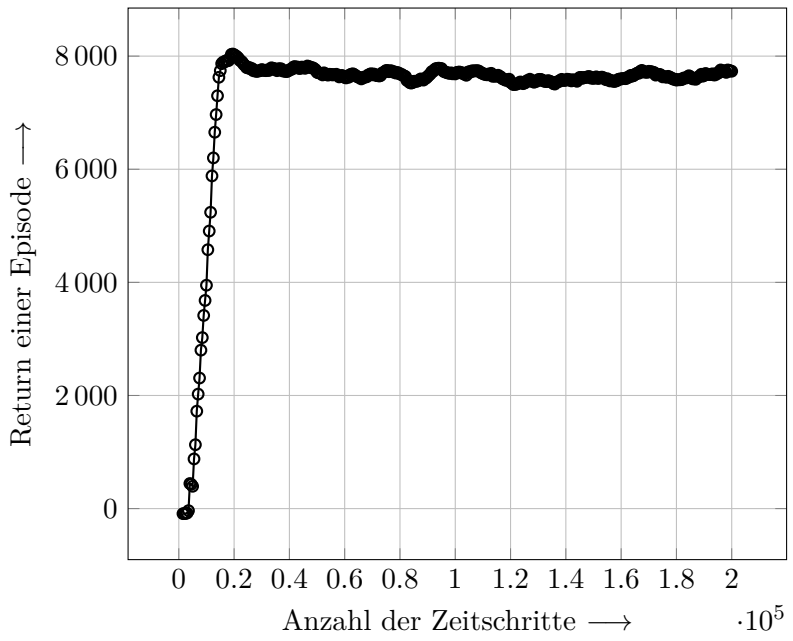


Abbildung 4.1.: Return des Standardmodells während des Trainings

Im Anschluss an die Trainingsphase können der trainierte Agent zur Regelung des Simulationsmodells in EcosimPro/ESPSS getestet und die erzielten Ergebnisse analysiert werden. Für die Auswertung wird ebenfalls eine Episodenlänge von 10 s gewählt. Da es sich bei dem untersuchten Fall um die Suche nach der Ventileinstellung für einen stationären Betriebspunkt handelt, ist diese Episodenlänge ausreichend. Nach einer Zeit von 10 s finden keine wesentlichen Änderungen der Betriebsparameter statt.

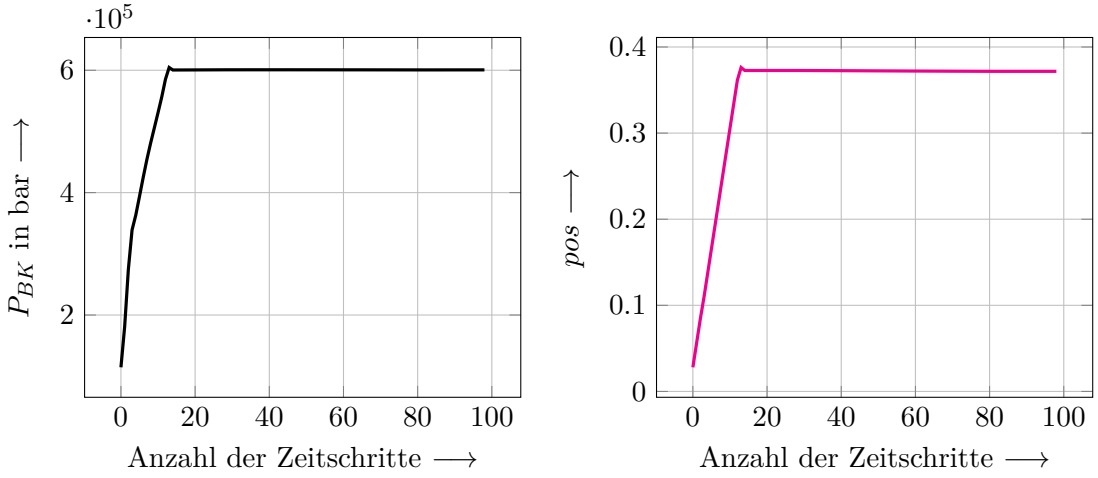


Abbildung 4.2.: Brennkammerdruck und Ventilposition des Standardmodells

In Abbildung 4.2 ist das Ergebnis des Brennkammerdrucks und der Regelventilposition über einen Betriebsdurchlauf in der Simulation aufgetragen. Das System befindet sich innerhalb der ersten Zeitschritte zunächst außerhalb des gewünschten Betriebspunktes. Der Regler fährt das Regelventil innerhalb der ersten 13 Zeitschritte auf eine Ventilstellung von 37 %. Nach umgerechnet 1,3 s erreicht der Brennkammerdruck den gewünschten Betriebspunkt von 6 bar

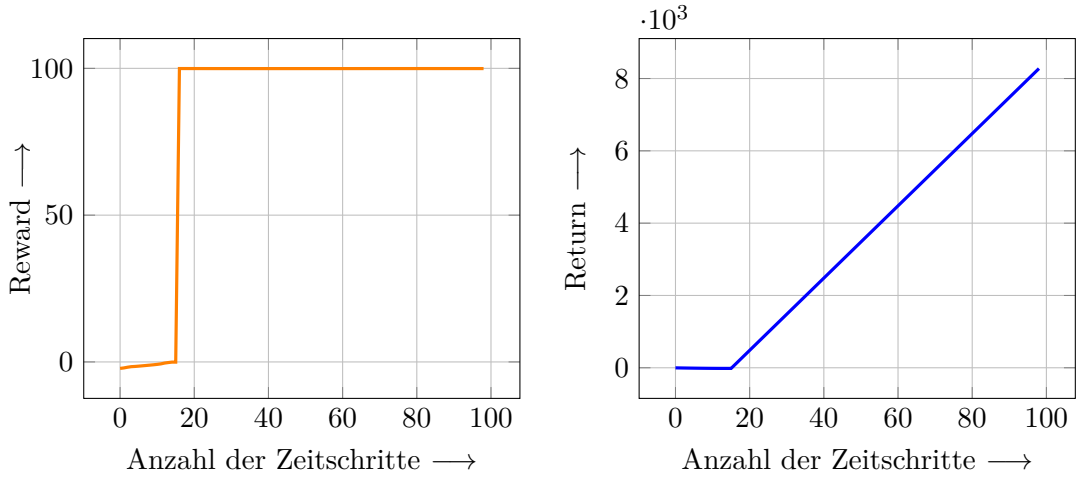


Abbildung 4.3.: Reward und Return des Standardmodells

Abbildung 4.3 zeigt, dass der Reward pro Zeitschritt zunächst aufgrund der Abweichungen des Initialzustandes negativ ist, innerhalb der ersten Zeitschritte allerdings auf einen Wert von 100 ansteigt. Somit wird der pro Zeitschritt maximal mögliche Reward erreicht. Das rechte Schaubild zeigt den aufsummierten Reward über alle Zeitschritte (Return). Dieser erreicht für das Standardmodell einen Wert von 8.276. In A3 kann die Notwendigkeit der in Abschnitt 3.3 beschriebenen Implementierung der Action-Funktion verdeutlicht werden. Ohne die Limitierung der Handlung auf ein maximales Öffnen von 5 % pro Zeitschritt kommt es zu Fluktuationen im Betriebspunkt, welche im Diagramm A3 anhand des Verlaufs der Regelventilposition deutlich gezeigt werden können. Die Ergebnisse des Standardmodells und des Modells mit der rudimentären Action-Funktion *simpleact* sind in 4.1 zusammengefasst. Neben der Zeit bis zum Erreichen des stationären Betriebspunkts t_{stat} ist die minimale ($\Delta_{st,min}$), maximale ($\Delta_{st,max}$) und mittlere ($\Delta_{st^*,mean}$) Abweichung vom Sollwert im stationären Zustand dokumentiert. Hier ist zu beachten, dass $\Delta_{st^*,mean}$ eine Vergleichsgröße ist, welche sich bei der Mittelung des Rewards über die Zeitschritte auf den Zeitpunkt des stationären Zustands des Standardmodells bezieht. Somit werden die ersten 13 Zeitschritte nicht für die Mittelwertbildung berücksichtigt. Zusätzlich wird in der Tabelle der Return angegeben.

Zeile zwei der Tabelle 4.1 zeigt die Ergebnisse des Standardmodells *std.1core*, welches auf einem statt auf 4 Kernen trainiert wurde. Es weist eine um 0,3 s langsamere Konvergenz des Brennkammerdrucks und einen niedrigeren Return auf.

Bezeichnung	t_{stat} [s]	$\Delta_{st,min}$ [Pa]	$\Delta_{st,max}$ [Pa]	$\Delta_{st^*,mean}$ [Pa]	Return [-]
Standardmodell	1,3	380	4520	688	8276
std.1core	1,6	20	6000	1667	7971
std.hold0	1,3	1360	2720	1820	8271
std.hold100	1,3	10	750	299	8279
std.hold5000	6,6	1700	7380	2176	2976
std.pos20perc	1,3	0	4070	616	8277
simpleact	1,3	50	2880	1393	8273

Tab. 4.1.: Vergleich der neuronalen Netze: Variation des Standardmodells

In den Zeilen 3-5 wurde der Effekt einer Ventilfixierung während der ersten Zeitschritte einer Episode getestet. Der Agent aus *std.hold100* greift nach 100 ms auf das Regelventil zu, bei *std.hold5000* geschieht dies erst nach 5000 ms, was 50 Zeitschritten entspricht. Vor diesem Zeitpunkt wird vom Agenten keine Handlung ausgeübt. Hiervon erhofft man sich ein zielgerichteteres Startverhalten für den Lernprozess. Das neuronale Netz *std.hold100* weist gegenüber *std.hold0* ein geringfügig genaueres Regelverhalten auf. Die Abweichungen von *std.hold5000* stammen aus der Blockierung der Ventilstellung innerhalb der ersten Episodenhälfte. Dadurch kann nur noch ein deutlich niedrigerer Returnwert als durch die anderen Agenten erreicht werden, welche das Regelziel bereits innerhalb der ersten Sekunden erfüllen.

Um zu überprüfen, ob die vorgegebene Ventilstellung zu Beginn einer Episode einen Einfluss auf das Ergebnis hat, wurde in *std.pos20perc* eine Initialöffnung für das Regelventil von 20 % vorgegeben. Das Standardmodell und alle weiteren trainierten neuronalen Netze verwenden eine Ventilposition von 80 %. Der Vergleich der Ergebnisse in Tabelle 4.1 weist keine Tendenz auf, dass die Ventilstellung zu Beginn einer Episode einen Einfluss auf die Ergebnisse hat.

4.1.1. Variation des Beobachtungsraums

In diesem Abschnitt wird analysiert, welche Größen des Observation-Space zielführend für die Erfüllung des Regelziels sind. Hierfür werden mit unterschiedlichen Beobachtungsräumen trainierte neuronale Netze miteinander verglichen. Im Folgenden sind alle untersuchten Parameter aufgelistet:

P_{BK}	Brennkammerdruck
P_{DR}	Druck im Rohr stromab des Druckreglers
P_{RV}	Druck im Rohr stromab des Regelventils
T_{RV}	Temperatur im Rohr stromab des Regelventils
$valvepos$	Ist-Wert des Regelventils
$valvepos'$	Ist-Wert der Regelventil-Öffnungsgeschwindigkeit
err	Abweichung des tatsächlichen Brennkammerdrucks vom Sollwert

Das Standardmodell verwendet die ersten 6 aufgelisteten Größen. Diese Wahl war in einem ersten Schritt bei der Findung eines geeigneten neuronalen Netzes als konservativ angesetzt. In einem weiteren Schritt soll nun untersucht werden, welche Größen des Beobachtungsraums für die Regelaufgabe ausreichend sind. Als zusätzliche Größe zu den bisher Beschriebenen wurde eine Fehlergröße *err* wie folgt implementiert:

$$err = \left|(P_{BK,Ist} - P_{BK,soll})\right| \tag{4.26}$$

Sie beschreibt die Abweichung des Ist-Zustandes des Brennkammerdrucks $P_{BK,Ist}$ vom Sollwert $P_{BK,soll}$, welcher 6 bar beträgt. Um einzugrenzen, welche der Parameter für das Erreichen des in dieser Arbeit definierten Regelziels notwendig sind, wurden sowohl Beobachtungsräume aus einzelnen Größen als auch aus Kombinationen mehrerer Größen getestet und mit dem Standardmodell verglichen. In Tabelle 4.2 sind die Ergebnisse entsprechend dem im vorigen Abschnitt eingeführten Schema aufgelistet. Die Abkürzung hinter der Bezeichnung *obs* weist auf die im Beobachtungsraum berücksichtigten Parameter hin. Die Tabelleneinträge mit der Kennzeichnung (x) weisen darauf hin, das keine entsprechenden Daten verfügbar sind, da das Regelziel nicht erreicht wurde und somit kein stationärer Zustand bestimmt werden kann. Einträge mit der Markierung (-) deuten an, dass die entsprechenden Daten aufgrund des in diesen Fällen eingeschränkten Beobachtungsraums nicht vollständig rausgeschrieben und eine vollständige Auswertung nicht durchgeführt werden konnte. Anhand des Returns ist jedoch ersichtlich, dass diese Konfigurationen das Regelziel erfolgreich erfüllen konnten.

Bezeichnung	t_{stat} [s]	$\Delta_{st,min}$ [Pa]	$\Delta_{st,max}$ [Pa]	$\Delta_{st^*,mean}$ [Pa]	Return [-]
Standardmodell	1,3	380	4520	688	8276
obs.BK	1,4	890	1600	924	8173
obs.RV	-	-	-	-	8279
obs.DR	x	x	x	x	-164
obs.T	x	x	x	x	-164
obs.pos	-	-	-	-	8275
obs.posvel	x	x	x	x	-159
obs.BK.pos	1,3	520	1800	986	8274
obs.BK.RV.DR	1,6	2290	2610	7965	8273
obs.err	-	-	-	-	7068
obs.std.err	1,3	50	920	713	8276

Tab. 4.2.: Vergleich der neuronalen Netze: Variation des Beobachtungsraums

Aus den Zeilen 4, 5 und 7 wird ersichtlich, dass ein Beobachtungsraum aus jeweils P_{DR}, T_{RV} und $valvepos'$ nicht zielführend ist. Mit den entsprechenden neuronalen Netzen wird lediglich ein negativer Return erzeugt. Alle weiteren Konfigurationen sind in der Regelung erfolgreich. Der Agent *obs.err* erreicht einen etwas geringeren Return von 7068 als die übrigen Varianten. Abbildung 4.4 veranschaulicht nochmals beispielhaft, dass ein

Observation-Space, welcher als einzige Größe den Brennkammerdruck P_{BK} verwendet, ein sehr ähnliches Ergebnis wie das Standardmodell erzeugt.

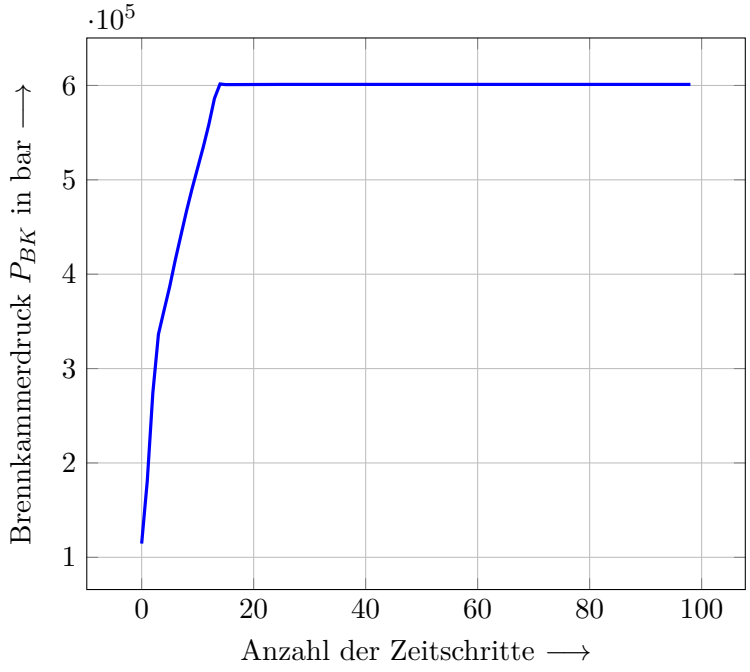


Abbildung 4.4.: Regelung des Brennkammerdrucks durch Agent obs.BK

Es lässt sich festhalten, dass der Brennkammerdruck dem Agenten bekannt sein muss, um eine Beziehung zwischen dem Reward, der von dem Sollwert abhängt, und den tatsächlichen Größen herstellen zu können. Dies kann entweder explizit über den Brennkammerdruck oder durch die in Gleichung 4.26 vorgestellte Formulierung einer Fehlergröße geschehen. Auch die tatsächliche Position des Regelventils ist wichtig um eine zeitliche Verzögerung zwischen Sollwertvorgabe und dem Erreichen des Sollwerts zu erkennen. Da der Druck P_{RV} in der Leitung zwischen Regelventil und Brennkammer ebenfalls zielführend war, kann vermutet werden, dass der Agent es schafft, hier einen Bezug zum Brennkammerdruck herzustellen, da die Leitung direkt in die Brennkammer mündet.

4.1.2. Test verschiedener Belohnungsfunktionen

Durch eine geeignete Formulierung der Reward-Funktion kann das Ergebnis einer RL-Anwendung maßgeblich verbessert werden. Schwankt beispielsweise der Brennkammerdruck um den angestrebten Sollwert, kann durch eine Bestrafung einer zu schnellen Bewegung der Regelventile das ungewollte Verhalten stabilisiert werden. Da die in dieser Arbeit formulierte Zielstellung der Regelung des Brennkammerdrucks auf einen stationären Sollwert bereits durch das Standardmodell mit der in Gleichung 3.22 beschriebenen Reward-Funktion erfüllt wurde, werden an dieser Stelle weitere Alternativen vorgeschla-

gen. Eine weitere Verbesserung der Regelung kann nicht erzielt werden, da diese bereits deutlich innerhalb der bestimmten Simulationsungenauigkeit liegt. In Tabelle 4.3 kann die mittlere Abweichung des Brennkammerdrucks vom Sollwert im stationären Zustand $\Delta_{st^*,mean}$ für das Standardmodell abgelesen werden. Diese beträgt 688 Pa, was etwa 7 % der geforderten Regelgenauigkeit von 0,1 bar entspricht. Eine weitere Verbesserung der Genauigkeit ist nicht zielführend. Auch die Zeit bis zum Erreichen des stationären Zustands t_{stat} kann durch die limitierte Ventilöffnung pro Aktion nicht mehr grundlegend verbessert werden.

Bezeichnung	t_{stat} [s]	$\Delta_{st,min}$ [Pa]	$\Delta_{st,max}$ [Pa]	$\Delta_{st^*,mean}$ [Pa]	Return [-]
Standardmodell	1,3	380	4520	688	8276
rew1	1,3	710	1960	1514	-28
rew1a.2	4,3	650	1480	5473	7387
rew1.2.3	1,3	2230	8090	2736	8287

Tab. 4.3.: Vergleich der neuronalen Netze: Variation der Belohnungsfunktion

Eine einfache Option bei der Formulierung einer Rewardfunktion ist die Verwendung der in Gleichung 3.23 beschriebenen *rew1*. Zeile 2 der Tabelle zeigt, dass hiermit ähnliche Ergebnisse wie mit dem Standardmodell erreicht werden können. Der Zahlenwert des Returns ist nicht mit dem des Standardmodells zu vergleichen. Der maximal erreichbare Return ist in diesem Fall ein Wert von 0. Abbildung 4.5 zeigt den Return der Rewardfunktion *rew1* während des Trainings.

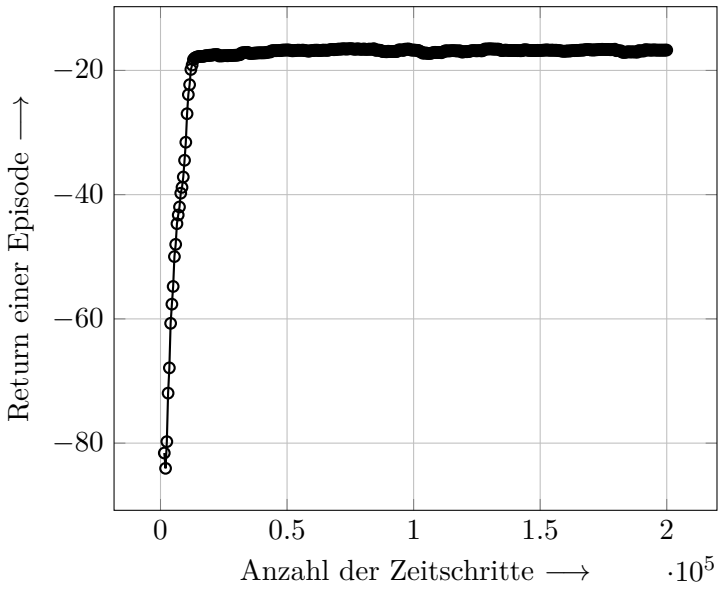


Abbildung 4.5.: Return der Rewardfunktion rew1 während des Trainings

Im Vergleich zur Standardfunktion fällt es in diesem Fall schwerer, einen Bezug zur Lösungsgüte herzustellen. Um auf die tatsächlichen Druckabweichungen zu schließen, muss der Reward gemäß Gleichung 3.23 in den Brennkammerdruck zurückgerechnet werden. Anhand *rew2* wird aufgrund der implementierten zusätzlichen Belohnung sofort ersichtlich, wann der stationäre Betriebspunkt erreicht ist und für wie viele Zeitschritte dieser Zustand eingehalten wird.

Das neuronale Netz *rew1a.2* verwendet im Gegensatz zum Standardmodell eine alternative Formulierung von *rew1*. Diese legt fest, dass der Anteil *rew1* in den ersten Sekunden der Episode keine Belohnung oder Bestrafung vergibt. Damit wird der Anfangszustand, in welchem die Ungenauigkeiten am höchsten sind, nicht für den Lernprozess berücksichtigt.

Die Rewardfunktion des Agenten *rew1.2.3* besitzt zusätzlich zu den im Standardmodell verwendeten Anteilen einen dritten Anteil *rew3*.

$$\begin{aligned} &\textbf{if} \quad valvepos' > 0 \\ &\qquad rew_3 = 1 \\ &\textbf{else} \\ &\qquad rew_3 = 0 \end{aligned} \tag{4.27}$$

Dieser Anteil bestraft die Bewegung des Regelventils pos' durch die Vergabe eines negativen Werts. Die Ergebnisse in 4.3 zeigen, dass diese Alternativen keinen großen Einfluss auf den hier vorliegenden Regelfall haben.

4.1.3. Domain Randomization und Noise

Für die Anwendung der Domain Randomization auf das in Kapitel 4.1 vorgestellte Standardmodell wird ein vereinfachtes Verfahren ähnlich der Automatic Domain Randomization [9] implementiert. Zu Beginn des Trainings findet zunächst keine Variation der Parameter statt. In dieser Phase wird somit am Standardmodell trainiert. Sobald ein vorher festgelegter Returnwert erreicht wird, startet die Variation eines vorher definierten Parameters in einer festgelegten Bandbreite. Dieser Vorgang kann beim erneuten Erreichen des festgelegten Returns wiederholt werden. Auf diese Weise können Level für Level unterschiedliche Variationsparameter in das Training einfließen und die Stabilität der Lösung erhöht werden, da der Agent lernt, auf nicht exakt vorgegebene Größen zu reagieren.

Die Parameter, welche für das hier verwendete Modell eines Orbitalantriebssystems in Frage kommen, können kategorisch unterschieden werden. Es gibt zum einen Größen, die den Druckverlust und damit das Strömungsverhalten des Simulationsmodells ändern. Hierzu zählen die Länge und Rauigkeit der Rohre, die Anzahl und Beschaffenheit der Rohrbiegungen und der Druckverlust am Injektor. Manche Parameter, wie beispielsweise der Umgebungsdruck und die Temperatur, ändern hingegen die Umgebungsbedingungen in der Simulation. Parameter wie die Ventilöffnungsgeschwindigkeit

beeinflussen die Systemcharakteristik. In dieser Arbeit wurden folgende Größen für die Domain Randomization implementiert:

P_{Vor}	In der Druckrandbedingung festgelegter Vordruck
L	Länge der Rohrleitung
$valvepos'$	Wert der Regelventil-Öffnungsgeschwindigkeit

Als Eingabewert müssen die Parameter, die variiert werden sollen, festgelegt werden. Dazu zählt auch die Bandbreite, in welcher die Größe schwanken soll. Aus Rechenzeit- und Performance-Gründen ist es sinnvoll, den Randomization-Bereich möglichst klein zu halten. Außerdem muss ein Grenzwert für den Return definiert werden, ab welchem das *Randomization-Level* erhöht wird. Für die hier durchgeführten Untersuchungen wurde dieser Returnwert auf 7.000 gesetzt. Dadurch kann sichergestellt werden, dass sich der Agent bereits für einige Zeitschritte im stationären Zustand befindet und die Lösung zu Beginn der Domain Randomization konvergiert.

Im ersten Level der DR wird der Vordruck P_{Vor} zufällig zwischen 75 bar und 85 bar variiert. Kann während des Trainings unter dieser Einstellung ein Return von 7000 erreicht werden, wird Level 2 aktiviert. Hier wird die Rohrlänge L im Bereich von 90 % bis 110 % der ursprünglichen Rohrlänge angepasst. Level 3 startet die Variation der Regelventil-Öffnungsgeschwindigkeit $valvepos'$ in einem Bereich von 0,2 bis 0,4.

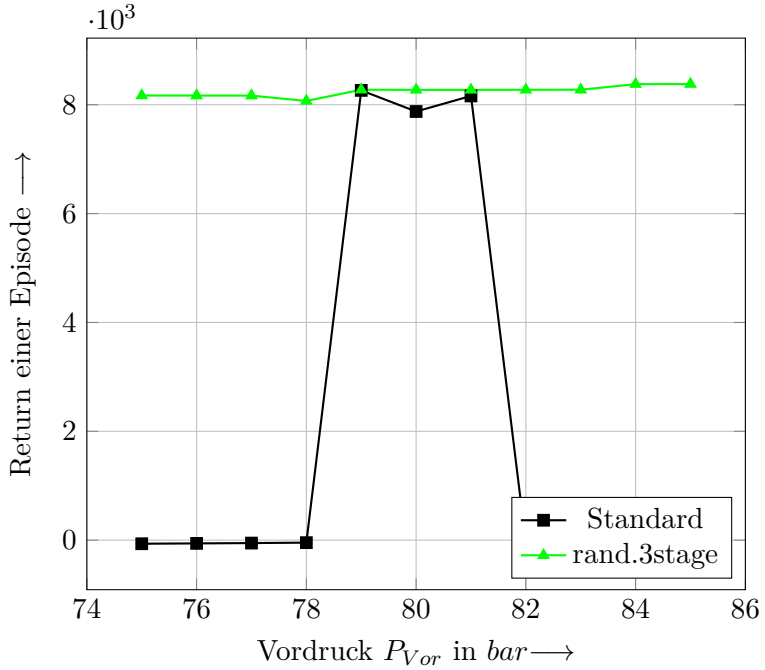


Abbildung 4.6.: Vergleich des Returns von DR und Standardmodell für verschiedene Vordrücke

Der Test des mit allen drei DR-Leveln trainierten Agenten *rand.3stage* zeigt in Abbildung 4.6, dass das mit DR trainierte neuronale Netz auch auf Vorgabewerte außerhalb

der im Modell festgelegten Größen mit einer gleichbleibenden Regelung reagieren kann. Dies wird aus dem Return des mit DR trainierten Agenten ersichtlich, welcher über das gesamte Spektrum des Vorgabedrucks konstant bleibt. Die Lösung des Standardmodells bricht hingegen ein, sobald der Vordruck um mehr als 1 bar vom in der Simulation spezifizierten Wert abweicht und erreicht außerhalb des Bereichs einen leicht negativen Return. Somit kann das Regelziel nicht weiter erfüllt werden.

Während des Trainings des Agenten *rand.3stage* fällt auf, dass insbesondere bei mit DR trainierten neuronalen Netzen der Lernprozess stark unterschiedlich ausfallen kann. Schaubild 4.7 zeigt den Unterschied im Return zweier mit exakt den gleichen Einstellungen spezifizierten Agenten. Dies ist darauf zurückzuführen, dass der Agent zu unterschiedlichen Zeitpunkten in das nächst höhere DR-Level startet. Dies hängt davon ab, wie erfolgreich die Exploration innerhalb der ersten Zeitschritte abläuft.

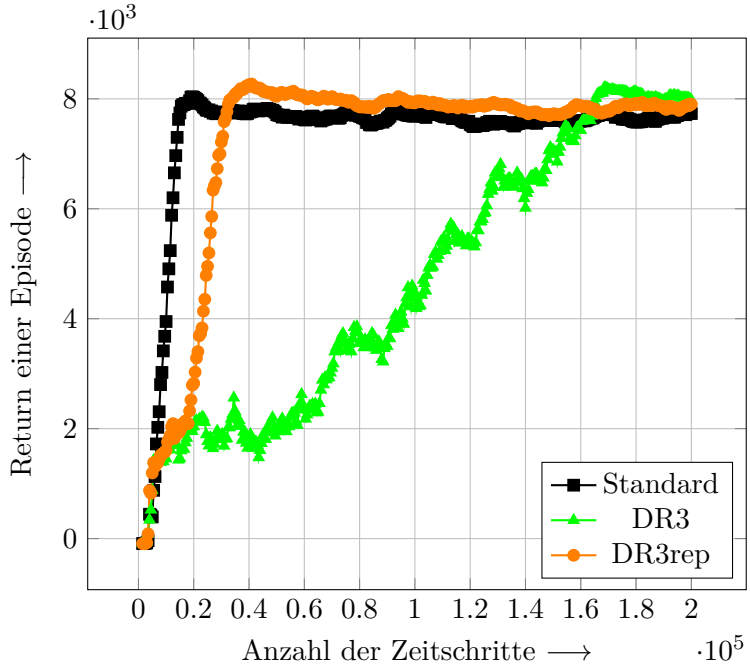


Abbildung 4.7.: Trainingsprozess von neuronalen Netzen mit Domain Randomization

Zusätzlich zu dem mit allen 3 Leveln trainierten Agenten werden weitere mit DR trainierte neuronale Netze miteinander verglichen. Hierfür werden die Agenten *rand.VD*, *rand.Rohr* und *rand.posvel* mit jeweils einem DR-Level der entsprechend variierten Größe trainiert. Außerdem wird in *rand.3stage.1000* und *rand.VD.1000* analysiert, wie sich die Lösungsgüte verändert, wenn bereits nach einem Return von 1000 die Domain Randomization aktiviert wird. Agent *rand.VD.posvel* stellt ein mit zwei Leveln trainiertes neuronales Netz dar. Die Ergebnisse sind in Tabelle 4.4 zusammengefasst.

Das Standardmodell ist als Referenz zu den mit DR trainierten Agenten angegeben. Es ist zu erkennen, dass für den hier vorliegenden Regelfall keine großen Unterschiede in der Lösungsgüte vorliegen. Die Returnwerte weichen maximal um einen Wert von

Bezeichnung	t_{stat} [s]	$\Delta_{st,min}$ [Pa]	$\Delta_{st,max}$ [Pa]	$\Delta_{st^*,mean}$ [Pa]	Return [-]
Standardmodell	1,3	380	4520	688	8276
rand.3stage	1,4	10	7690	1007	8175
rand.3stage.rep	1,3	10	830	380	8277
rand.3stage.1000	1,3	950	4900	1517	8273
rand.VD	1,3	810	1660	1357	8272
rand.VD1000	1,3	1610	5040	2002	8271
rand.Rohr	1,7	1240	6750	1797	7864
rand.posvel	1,3	0	1300	406	8278
rand.VD.posvel	1,3	0	1730	387	8278

Tab. 4.4.: Vergleich der neuronalen Netze: Variation der Domain Randomization

100 voneinander ab, was lediglich einem Zeitschritt mehr oder weniger im stationären Zustand pro Episode entspricht. Eine Ausnahme davon stellt *rand.Rohr* dar. Der Return dieses neuronalen Netzes ist mit 7.864 um etwa 400 niedriger als der des Standardmodells. Auch der stationäre Zustand wird erst 0,4 s später erreicht. Allerdings ist auch hier das Regelziel erreicht worden.

Als Alternative zur Domain Randomization, die bestimmte Parameter nach einer Episode im Trainingsprozess variiert, kann stattdessen eine Variation der Parameter auch innerhalb einer Episode stattfinden. Beim Einbringen von Störgrößen innerhalb einer Episode kann von Rauschen (eng. *Noise*) gesprochen werden.

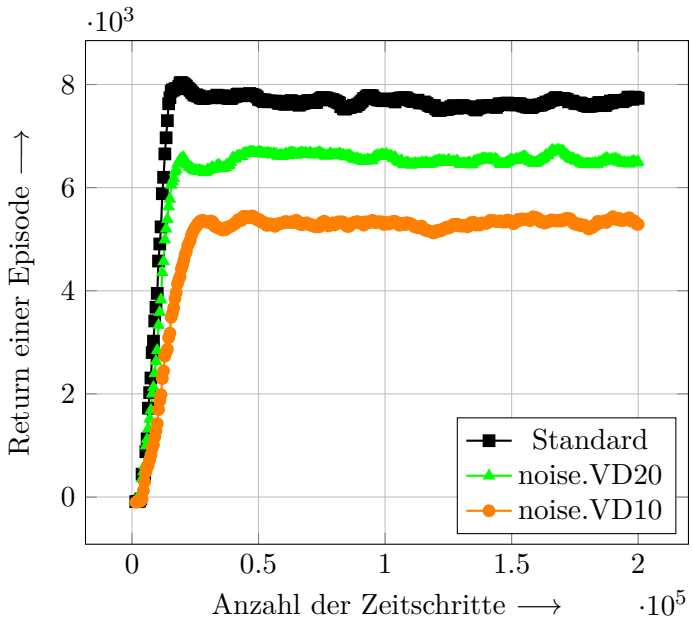


Abbildung 4.8.: Trainingsprozess der Agenten noise.VD20 und noise.VD10

Anstatt der DR-Level wird bei dieser Methode festgelegt, in welchen Zeitschritt-Abständen eine Schwankung einer vorher definierten Größe in das Training einbezogen wird. Im Folgenden werden die gleichen Parameter wie in der eben vorgestellten Domain Randomization variiert. Dazu zählen der Vordruck und die Ventilöffnungsgeschwindigkeit. Die Bandbreite bleibt bestehen. Abbildung 4.8 zeigt den Trainingsprozess von zwei mit Noise trainierten Agenten im Vergleich zum Standardmodell. Das neuronale Netz *noise.VD20* wurde mit einer Schwankung im Vordruck, welche alle 20 Zeitschritte variiert wird, trainiert. Dazu wurde die Störgröße in *noise.VD10* alle 10 Zeitschritte eingebracht. Wie oben beschrieben, entspricht die Variation der Bandbreite der in der Domain Randomization festgelegten Definition. Anhand des Verlaufs des Returns ist zu erkennen, dass dieser gegen einen geringeren Wert konvergiert, je öfter eine Störung in die Rechnung eingebracht wird. Um zu vermeiden, dass ein zu niedrigerer Return für die erfolgreiche Regelung des Brennkammerdrucks erreicht wird, sollte die Variation der Störgröße nicht in zu geringen Abständen der Zeitschritte stattfinden.

Abbildung 4.9 zeigt das Ergebnis von Agent *noise.VD10* für die Regelung des Brennkammerdrucks. Hier ist anhand des Verlaufs des Graphen die Störschwankung zu erkennen, welche alle 10 Zeitschritte auftritt. Wird das Rauschen im Testprozess deaktiviert, erreicht der Agent eine ähnliches Ergebnis wie das Standardmodell. Ein Vergleich dazu ist in Tabelle 4.5 festgehalten.

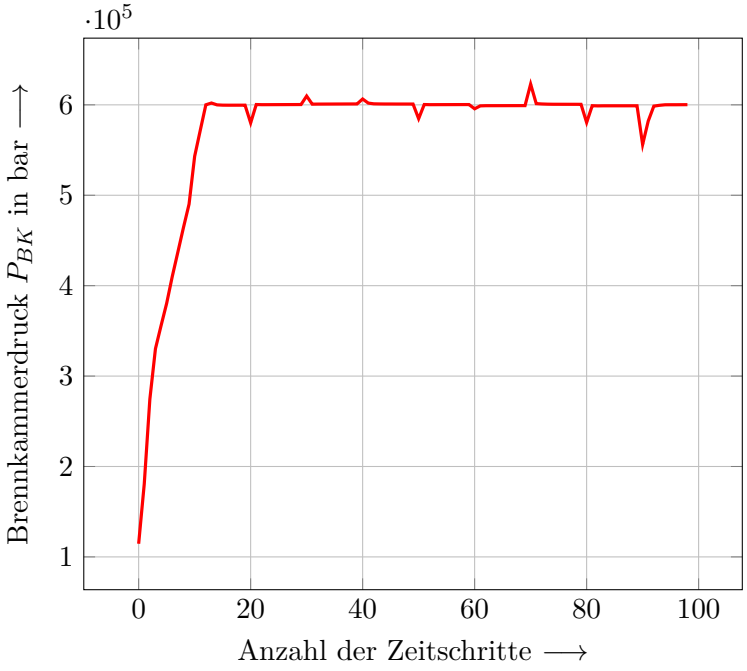


Abbildung 4.9.: Regelung des Brennkammerdrucks mit Agent noise.VD10

Der Vorteil eines mit Noise trainierten neuronalen Netzes wird aus Schaubild 4.10 deutlich. Ähnlich wie bei der Domain Randomization können auch bei Abweichung vom Vordruck von 80 bar gleiche Returnwerte erzielt werden. Bei dem mit Noise trainierten

Agenten konnte sogar noch bei einem Vordruck von 100 bar ein relativ hoher Return von 7.871 erreicht werden. Der letzte positive Return des mit DR trainierten Agenten liegt bei einem Vordruck von 98 bar. Agent *noise.VD20* erreicht bis zu einem Vordruck von 131 bar den letzten positiven Reward. Das bedeutet, dass der Agent selbst bei einer Abweichung des Vordrucks von 51 bar das Regelziel in zumindest einem Zeitschritt erfüllen konnte, was die enorme Robustheit des Verfahrens gegenüber dem Standardmodell verdeutlicht.

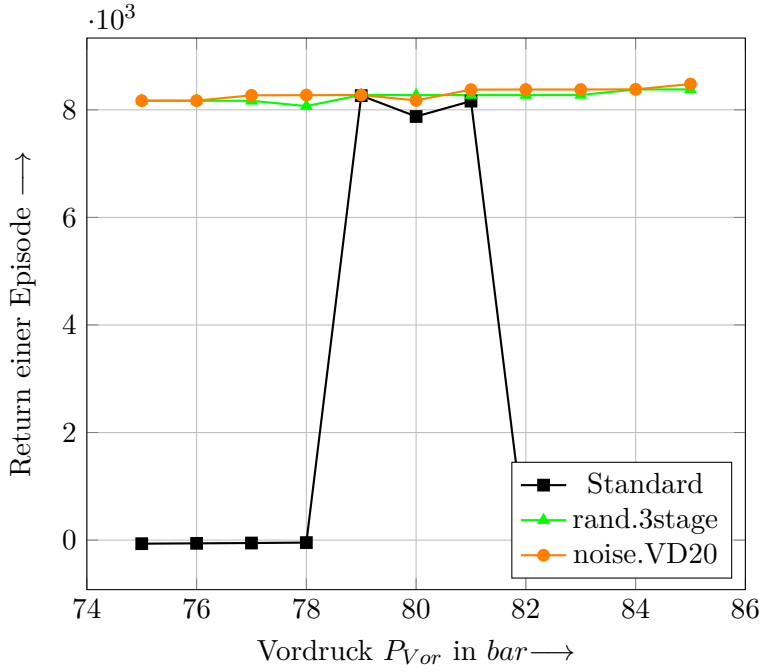


Abbildung 4.10.: Vergleich des Returns von DR und Noise für verschiedene Vordrücke

Alle erzielten Ergebnisse der mit Noise trainierten neuronalen Netze sind in Tabelle 4.5 zusammengefasst. Zu den bisher erläuterten Agenten sind hier *noise.posvel20* und *noise.VD.posvel20* ausgewertet.

Bezeichnung	t_{stat} [s]	$\Delta_{st,min}$ [Pa]	$\Delta_{st,max}$ [Pa]	$\Delta_{st^*,mean}$ [Pa]	Return [-]
Standardmodell	1,3	380	4520	688	8276
noise.VD10	1,2	70	1410	832	6273
noise.VD20	1,4	870	1000	1302	6870
noise.posvel20	1,3	10	7510	411	8277
noise.VD.posvel20	1,3	80	9760	1347	7371

Tab. 4.5.: Vergleich der neuronalen Netze: Variation der Störgrößen (*Noise*)

In *noise.posvel20* wurde die Ventilöffnungsgeschwindigkeit alle 20 Zeitschritte innerhalb einer Episode variiert. Der Agent *noise.VD.posvel20* vereint die zwei Störgrößen Vordruck und Ventilöffnungsgeschwindigkeit. Alle Agenten erfüllen das Regelziel. Das neuronale Netz *noise.VD10* erreicht den stationären Zustand am schnellsten von allen in dieser Arbeit untersuchten Agenten. Agent *noise.posvel20* erreicht einen fast identischen Return wie das Standardmodell. Dies hängt damit zusammen, dass der Druck, welcher für die Definition der Rewardfunktion herangezogen wurde, nicht variiert wird. Der leicht verminderte Return der restlichen Agenten lässt sich mit der Einbringung der Störung des Vordrucks erklären, welche einen größeren Einfluss auf den Brennkammerdruck hat als die Ventilöffnungsgeschwindigkeit.

4.1.4. Berücksichtigung von Ventilverzögerung

Da am realen Prüfstand eine Verzögerung zwischen Stellsignal an das Regelventil und der Ausführung der entsprechenden Ventilbewegung vorhanden ist, wird versucht, diese Eigenschaft auch beim Training der neuronalen Netze zu berücksichtigen. Die auftretende Verzögerung (eng. *Delay*) beträgt für das Regelventil in der Realität etwa 300 ms. Für die Implementierung in Python geschieht dies dadurch, dass die jeweilig zu einem Zeitschritt berechnete Aktion erst drei Zeitschritte später an das Regelventil übergeben wird. Da drei Zeitschritte umgerechnet einer Zeit von 300 ms entsprechen, kann auf diese Weise eine Delay-Funktionalität gewährleistet werden. Die in diesem Abschnitt vorgestellten neuronalen Netze verwenden diese Umsetzung des Delays für den Trainingsprozess und die Tests in der Simulation.

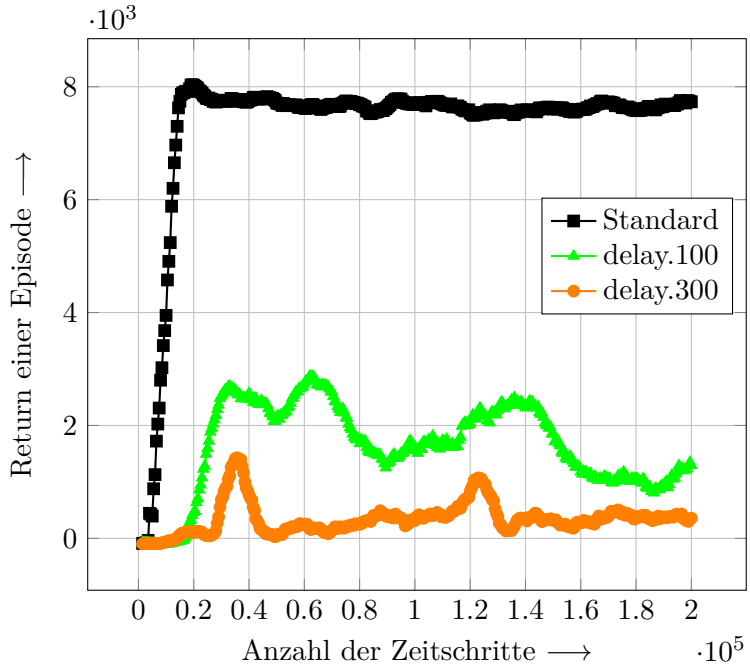


Abbildung 4.11.: Trainingsprozess der Agenten mit Berücksichtigung von Delay

Der Verlauf des Returns der Agenten *delay.100* und *delay.300* weist einen erheblich niedrigeren Wert als der des Standardmodells auf. Eine Konvergenz ist nach 200.000 Zeitschritten noch nicht zu erkennen. Agent *delay.300* erreicht zum Teil einen negativen Reward. Es kann davon ausgegangen werden, dass der Agent Probleme bei der Regelung das Brennkammerdrucks haben wird. Dies wird auch im Test des Agenten in der Simulation in Abbildung 4.12 anhand der blauen Verlaufslinie bestätigt. Der Versuch, die Regelung durch ein mit Noise trainiertes neuronales Netz *d.noise.VD20* zu stabilisieren, gelingt nicht. Die Tendenz geht in Richtung des richtigen Brennkammerdrucks, allerdings weisen beide Agenten, welche mit einem Delay von 300 ms trainiert wurden, deutliche Schwankungen auf.

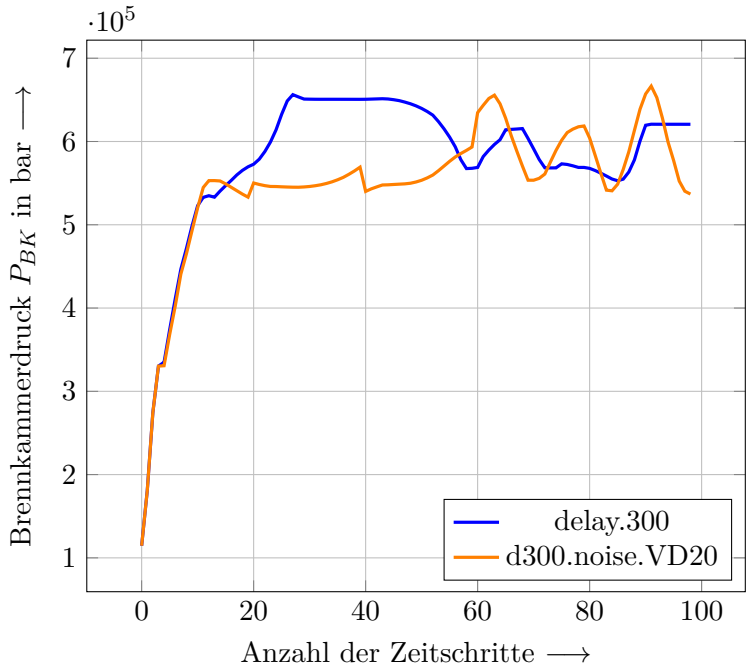


Abbildung 4.12.: Test der Agenten delay.300 und d300.noise.VD20 in der Simulation

Es bleibt zu überprüfen, ob der RL-Agent prinzipiell unter Berücksichtigung von Delay funktionieren kann. Um dies zu klären, werden Agenten getestet, welche mit einem geringeren Delay von 100 ms trainiert wurden.

In Abbildung 4.13 kann gezeigt werden, dass beide Agenten das Regelziel erfüllen können. Agent *delay.100* benötigt 5,7 s bis zum Erreichen des stationären Betriebspunkts. Der mit DR trainierte Agent *d100.rand.VD* ist mit 3,2 s etwas schneller, weist jedoch innerhalb der ersten 60 Zeitschritte Schwankungen auf. Generell kann festgehalten werden, dass die Berücksichtigung von Verzögerungen der Aktion des Agenten eine Herausforderung für RL-Prozesse darstellt. Allerdings muss an dieser Stelle infrage gestellt werden, ob eine Berücksichtigung von Delay zur Regelung der realen Anwendung überhaupt notwendig ist.

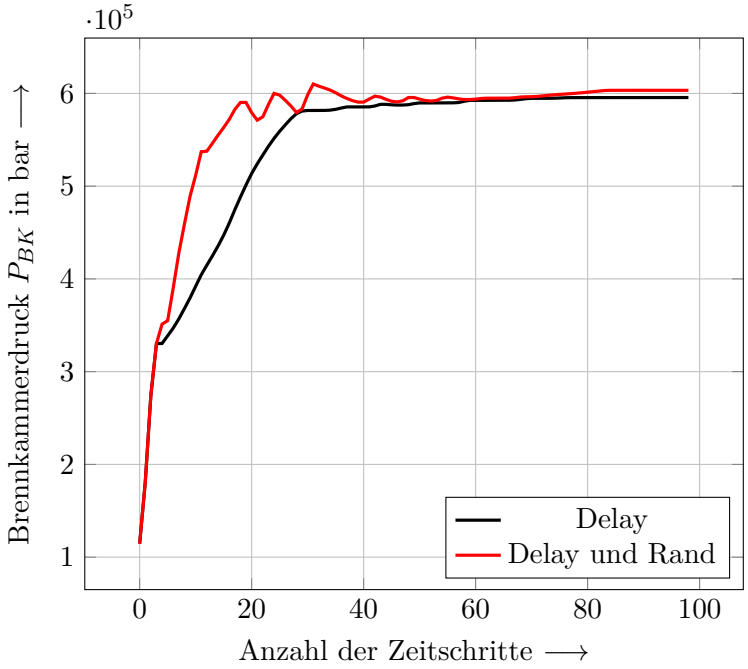


Abbildung 4.13.: Test der Agenten delay.100 und d100.rand.VD in der Simulation

Alle Ergebnisse der mit Delay trainierten Agenten sind in Tabelle 4.6 zusammengefasst. Der Return des Standardmodells kann in keiner Variante erreicht werden. Das neuronale Netz *d100.r.n.VD20* konnte keine zielführende Regelung erreichen. Bei diesem Agenten wurde sowohl DR als auch Noise im Trainingsprozess berücksichtigt. Auch die mit einem Delay von 300 ms trainierten Agenten scheiterten an der Regelung des stationären Betriebspunkts in der Simulation.

Bezeichnung	t_{stat} [s]	$\Delta_{st,min}$ [Pa]	$\Delta_{st,max}$ [Pa]	$\Delta_{st^*,mean}$ [Pa]	Return [-]
Standardmodell	1,3	380	4520	688	8276
delay.300	x	x	x	x	-65
d300.noise.VD20	x	x	x	x	-71
delay.100	5,7	4440	9810	23761	3844
d100.rand.VD	3,2	210	9400	5544	6361
d100.r.n.VD20	x	x	x	x	-164

Tab. 4.6.: Vergleich der neuronalen Netze: Variation der Ventilverzögerung (*Delay*)

4.2. Erprobung ausgewählter Regler am Prüfstand

Im finalen Schritt dieser Masterarbeit werden ausgewählte neuronale Netze im realen Prüfstandsversuch erprobt. Der Prüfstandsaufbau ist in Abschnitt 3.1 beschrieben. Für den Triebwerksbetrieb wird eine vorgegebene Steuersequenz verwendet. Für die hier durchgeführten Testfälle beträgt die Dauer der gesamten Sequenz 34 s.

Eine Sekunde nach Sequenzstart beginnt die Datenspeicherung, eine weitere Sekunde später wird der Druckregler und die Regelventilposition eingestellt. Nach 5 s öffnet sich das Schussventil. Ab Sekunde 10 startet die Regelung durch die KI. Diese bleibt über eine Versuchsdauer von 20 s aktiv. Eine Sekunde nach Beendigung des Versuchs wird das Schussventil wieder geschlossen. Nach zwei weiteren Sekunden beginnt das Entlüften des Druckreglers und nach insgesamt 34 s wird die Datenspeicherung schließlich beendet.

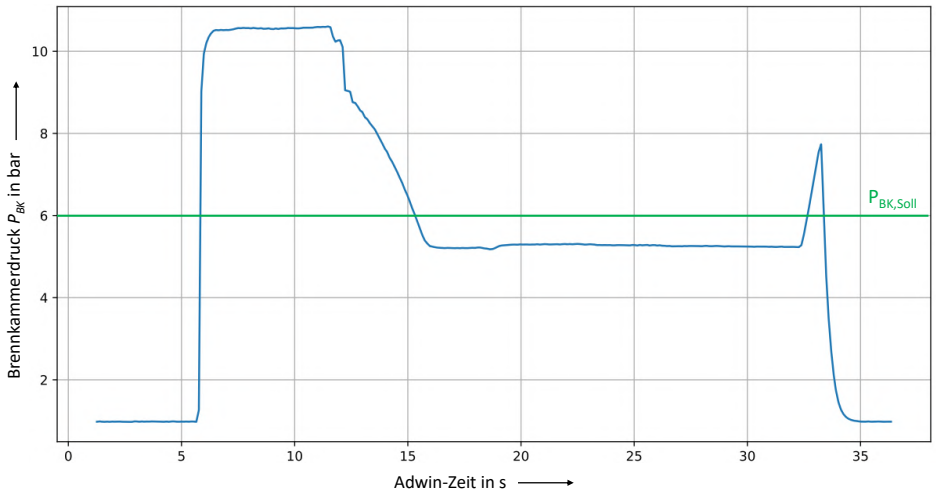


Abbildung 4.14.: Erster Prüfstandsversuch mit RL basierter Regelung durch das Standardmodell

In Abbildung 4.14 wird das Ergebnis eines ersten Regelversuchs durch das Standardmodell vorgestellt. Zunächst fällt auf, dass eine gewisse Verzögerung zwischen der in der Sequenz vorgegebenen Steuereingaben und den im Versuch aufgezeichneten Reaktionen darauf existiert. Der Druck steigt somit erst nach etwas mehr als 5 s an. Nach etwa 12 s kann der Effekt der KI-Regelung beobachtet werden. Das Regelziel kann in diesem ersten Test mit dem Standardmodell nicht erreicht werden. Der Brennkammerdruck wird hier auf einen stationären Wert von 5,2 bar geregelt und liegt somit außerhalb der geforderten minimalen Abweichung von 0,1 bar vom Sollwert. Der Sollwert ist im Diagramm in grün eingezeichnet und beträgt wie bereits beschrieben 6 bar.

Um das Regelergebnis am Prüfstand zu verbessern, wurden weitere neuronale Netze trainiert, welche insbesondere den Fehler *err* im Observation-Space berücksichtigen. Die bisher gesammelten Erfahrungen am Prüfstand konnten zeigen, dass auf diese Weise ein besseres Regelergebnis erzeugt werden kann. Daher sind für die weiteren Testversuche

Bezeichnung	t_{stat} [s]	$\Delta_{st,min}$ [Pa]	$\Delta_{st,max}$ [Pa]	$\Delta_{st^*,mean}$ [Pa]	Return [-]
Standardmodell	1,3	380	4520	688	8276
std.err.delay300	x	x	x	x	-70
std.err.r.VD2k	1,8	830	5560	3344	7768
std.err.r.VD2k.a	1,3	580	4350	1096	8274
std.err.r.VD2k.a.Rohr	1,3	680	3640	1392	8273
std.err.r.VD2k.a.d100	x	x	x	x	-53

Tab. 4.7.: Vergleich der für den Prüfstandsversuch trainierten neuronalen Netze

die in Tabelle 4.7 aufgeführten Agenten trainiert und zunächst in der Simulation getestet und ausgewertet worden. Die Varianten in Zeile 2 und 6 berücksichtigen den Delay und konnten in der Simulation nicht das Regelziel erfüllen. Agent *std.err.r.VD2k* basiert auf dem Standardmodell und verwendet zusätzlich den Fehler *err* im Beobachtungsraum. Hier wurde mit Domain Randomization des Vordrucks trainiert, nachdem für den Return ein Schwellwert von 2.000 erreicht wurde. Das neuronale Netz *std.err.r.VD2k.a* ist mit identischen Einstellungen wie *std.err.r.VD2k* trainiert worden. Lediglich für die Domain Randomization wurde die Bandbreite des Vordrucks auf 60-90 bar erweitert. Agent *std.err.r.VD2k.a.Rohr* berücksichtigt zusätzlich ein weiteres Level der Domain Randomization, in welchem die Rohrlängen variiert werden.

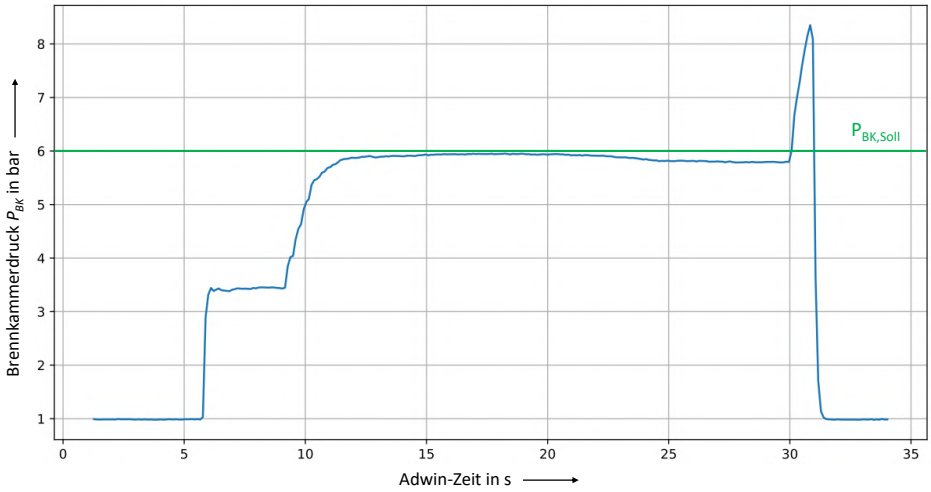


Abbildung 4.15.: Prüfstandsversuch mit RL basierter Regelung durch Agent std.err.r.VD2k

Am geeignetsten für die Regelung des realen Prüfstands zeigte sich Agent *std.err.r.VD2k*. Das Ergebnis der Regelung des Brennkammerdrucks ist in Abbildung 4.15 aufgetragen. Der Verlauf der blauen Kurve zeigt, dass der Regler das Regelziel etwa 5 s nach Aktivierung der KI-Regelung in der Regelsequenz erreicht. Der stationäre

Zustand kann 5 s eingehalten werden. Anschließend sinkt der Brennkammerdruck leicht ab und verlässt die geforderte minimale Abweichung von 0,1 bar vom Sollwert.

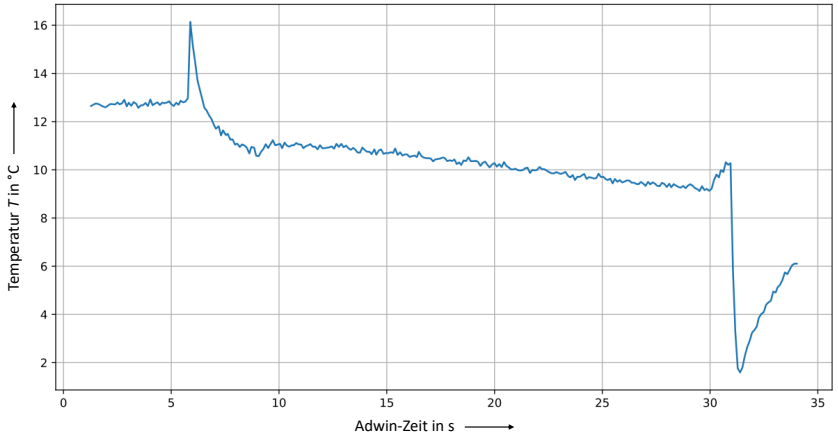


Abbildung 4.16.: Temperaturverlauf während des Prüfstandbetriebs

Eine mögliche Erklärung für dieses Verhalten ist das Absinken der Temperatur während des Prüfstandbetriebs mit Kaltgas. Insbesondere wenn mehrere Versuche nacheinander gefahren werden, sinkt die Temperatur der Komponenten ab. Somit herrschen nicht dieselben Umgebungsbedingungen wie in der Simulation. Der Verlauf der Temperatur während des Prüfstandtests ist in Abbildung 4.16 dargestellt.

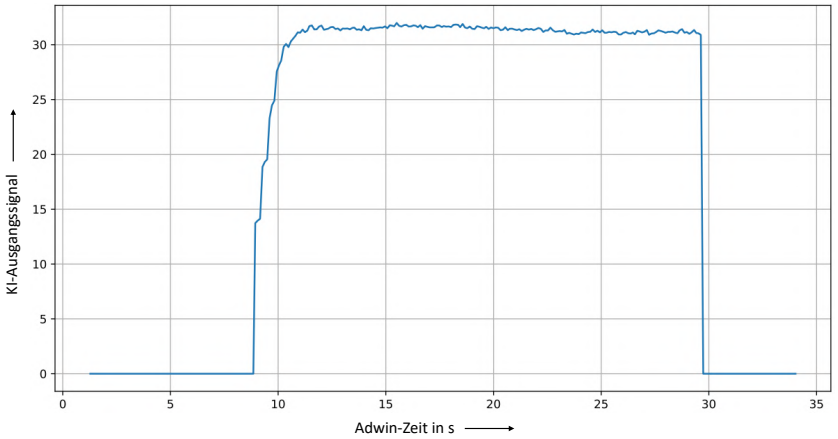


Abbildung 4.17.: Nachweis des KI-Ausgangssignals für den Prüfstandsversuch

Die prinzipielle Arbeitsweise der KI-Regelung kann in Abbildung 4.17 nachgewiesen werden. Entsprechend der beschriebenen Steuersequenz arbeitet der Regler in dem gewollten Zeitabschnitt und wird ab Sekunde 30 abgestellt. Die Daten für das Schaubild des KI-Steuersignals stammen wie die Temperaturdaten ebenfalls aus dem mit Agent

std.err.r.VD2k durchgeführten Prüfstandversuch. Die übrigen Agenten aus Tabelle 4.7 konnten das Regelziel nicht erfüllen. Variante *std.err.r.VD2k.a* konnte in dieser Arbeit noch nicht im Prüfstandsversuch getestet werden. Mit den hier aufgeführten Ergebnissen konnte nachgewiesen werden, das ein RL basierter Regler einen stationären Betriebspunkt für Raketentriebwerk auch am realen Prüfstand einstellen kann. Es ist anzumerken, dass während dieser Arbeit Umbauten am Prüfstand stattgefunden haben. Diese wurden nach der Validierung des Simulationsmodells durchgeführt. Die genauen Änderungen und deren Einfluss auf das hier verwendete Modell sind bislang nicht bekannt. Es ist zu vermuten, dass dieser Umstand die Lösungsgüte der Regelung am realen Prüfstand beeinflusst hat. Dennoch konnte ein neuronales Netz trainiert werden, welches das in dieser Arbeit gesetzte Ziel der Regelung eines stationären Betriebszustands für einen realen Prüfstandsversuch in weiten Teilen erfüllt. Dies lässt auf die Robustheit von RL basierten Regelverfahren schließen.

5. Zusammenfassung und Ausblick

Für den Test eines RL basierten Reglers in der Realität ist es zweckmäßig, ein möglichst einfaches System für erste Versuche in der realen Anwendung aufzubauen. Ein Beispiel dafür ist das hier vorgestellte Modell eines Stickstoff-Kaltgasantriebs mit einem Regelventil. Anhand eines solchen Aufbaus können die Effekte und Probleme, welche beim Übertragen von der Simulation in die Realität auftreten, näher betrachtet werden.

In dieser Masterarbeit wurde ein EcosimPro/ESPSS-Simulationsmodell des Green Propellant Prüfstands am M11.5 des Deutschen Zentrums für Luft- und Raumfahrt e.V. erstellt. Das Modell ist mit Messdaten des realen Prüfstands validiert worden. Mit dem Reinforcement Learning Framework *Ray* wurden Reinforcement Learning basierte Regler trainiert, die es ermöglichen, den Brennkammerdruck des Triebwerks in der Simulation und im realen Versuch zu regeln. Die in der Arbeit vorgestellten Regler verwenden den *Soft Actor-Critic*-Algorithmus.

Bei der Übertragung vom Simulationsmodell auf den Prüfstand werden Regelverfahren benötigt, die bezüglich der Unterschiede zwischen Simulation und realer Anwendung robust sind und weiterhin zuverlässig arbeiten. Um ein geeignetes neuronales Netz für die Regelung eines stationären Betriebspunkts des Brennkammerdrucks zu finden, wurden intensive Studien verschiedener Parameter durchgeführt und insgesamt 42 Agenten trainiert. Diese Agenten wurden im Anschluss an das Training in der Simulation getestet. Eine Übersicht aller Ergebnisse ist in Tabelle 5.1 aufgeführt.

Es wurde der Beobachtungsraum für den gegebenen Anwendungsfall untersucht und verschiedene Belohnungsfunktionen implementiert und getestet. Um die Robustheit des Reglers zu erhöhen, wurde die Methode der *Domain Randomization* implementiert und angewendet. Zusätzlich wurden Störgrößen in den Trainingsprozess eingebracht und der Einfluss von Ventilverzögerungen auf die Regelgüte untersucht.

Im Rahmen der Arbeit konnte nachgewiesen werden, dass eine Großteil der trainierten neuronalen Netze das Regelziel innerhalb einer Simulationsumgebung erfüllen kann. Die Regelung des realen Prüfstands stellte für die meisten Regler eine Herausforderung dar. Insbesondere die Berücksichtigung einer Fehlergröße für den Brennkammerdruck im Beobachtungsraum und die Anwendung von Domain Randomization haben einen entscheidenden Einfluss auf den Regelerfolg in der realen Anwendung gemacht. Ein mit diesen Einstellungen trainierter RL basierter Regler konnte für den in dieser Arbeit beschriebenen Modellaufbau bereits sehr gute Ergebnisse für verschiedene Betriebspunkte erzielen [49]. Die Ergebnisse wurden neulich auf der *8th Space Propulsion Conference* in Portugal präsentiert.

Die durch das Training der in Tabelle 5.1 dargestellten neuronalen Netze gesammelten Erfahrungen dienen der zukünftigen Entwicklung RL basierter Regler als Erkenntnisgewinn. Insbesondere die Domain Randomization und die Einbringung von Störgrößen

Bezeichnung	t_{stat} [s]	$\Delta_{st,min}$ [Pa]	$\Delta_{st,max}$ [Pa]	$\Delta_{st^*,mean}$ [Pa]	Return [-]
Standardmodell	1,3	380	4520	688	8276
std.1core	1,6	20	6000	1667	7971
std.hold0	1,3	1360	2720	1820	8271
std.hold100	1,3	10	750	299	8279
std.hold5000	6,6	1700	7380	2176	2976
std.pos20perc	1,3	0	4070	616	8277
simpleact	1,3	50	2880	1393	8273
obs.BK	1,4	890	1600	924	8173
obs.RV	-	-	-	-	8279
obs.DR	x	x	x	x	-164
obs.T	x	x	x	x	-164
obs.pos	-	-	-	-	8275
obs.posvel	x	x	x	x	-159
obs.BK.pos	1,3	520	1800	986	8274
obs.BK.RV.DR	1,6	2290	2610	7965	8273
obs.err	-	-	-	-	7068
obs.std.err	1,3	50	920	713	8276
rew1	1,3	710	1960	1514	-28
rew1a.2	4,3	650	1480	5473	7387
rew1.2.3	1,3	2230	8090	2736	8287
rand.3stage	1,4	10	7690	1007	8175
rand.3stage.rep	1,3	10	830	380	8277
rand.3stage.1000	1,3	950	4900	1517	8273
rand.VD	1,3	810	1660	1357	8272
rand.VD1000	1,3	1610	5040	2002	8271
rand.Rohr	1,7	1240	6750	1797	7864
rand.posvel	1,3	0	1300	406	8278
rand.VD.posvel	1,3	0	1730	387	8278
noise.VD10	1,2	70	1410	832	6273
noise.VD20	1,4	870	1000	1302	6870
noise.posvel20	1,3	10	7510	411	8277
noise.VD.posvel20	1,3	80	9760	1347	7371
delay.300	x	x	x	x	-65
d300.noise.VD20	x	x	x	x	-71
delay.100	5,7	4440	9810	23761	3844
d100.rand.VD	3,2	210	9400	5544	6361
d100.r.n.VD20	x	x	x	x	-164
std.err.delay300	x	x	x	x	-70
std.err.r.VD2k	1,8	830	5560	3344	7768
std.err.r.VD2k.a	1,3	580	4350	1096	8274
std.err.r.VD2k.a.Rohı	1,3	680	3640	1392	8273
std.err.r.VD2k.a.d100	x	x	x	x	-53

Tab. 5.1.: Vergleich der Simulationsergebnisse aller trainierten neuronalen Netze

innerhalb einer Episode während des Trainingsprozesses sind vielversprechend in Bezug auf die Erhöhung der Robustheit. Somit sind in Zukunft auch komplexere Regelziele denkbar. Beispiele hierfür sind die Effizienzoptimierung eines Triebwerks bei gegebenem Schub oder die Minimierung von Wandtemperaturen im Triebwerksbetrieb [57].

Literatur

[1] S. Colas u. a. „A point of view about the control of a reusable engine cluster“. In: *Proceedings of the 8th European Conference for Aeronautics and Space Sciences*. 2019.

[2] P. Tatiossian, J. Desmariaux und M. Garcia. „A point of view about the control of a reusable engine cluster“. In: *Proceedings of the 7th European Conference for Aeronautics and Space Sciences*. 2017.

[3] Kai Dresia, Günther Waxenegger-Wilfing und Michael Oschwald. „Nonlinear Control of an Expander-Bleed Rocket Engine using Reinforcement Learning“. In: *Proceedings of the Space Propulsion 2020+1 Conference, Virtual Event* (2021).

[4] Günther Waxenegger-Wilfing u. a. *A Reinforcement Learning Approach for Transient Control of Liquid Rocket Engines*. 2020. arXiv: 2006.11108 [cs.LG].

[5] Till Hörger. *Reinforcement Learning Framework for Optimal Control of Orbital Propulsion considering Systems Robustness and Operating Limitations*. Deutsches Zentrum für Luft- und Raumfahrt, Institut für Raumfahrtantriebe, 2021.

[6] Günther Waxenegger-Wilfing u. a. *Machine Learning Methods for the Design and Operation of Liquid Rocket Engines – Research Activities at the DLR Institute of Space Propulsion*. 2021.

[7] Volodymyr Mnih u. a. *Playing Atari with Deep Reinforcement Learning*. 2013. arXiv: 1312.5602 [cs.LG].

[8] Christopher Berner u. a. *Dota 2 with Large Scale Deep Reinforcement Learning*. 2019.

[9] OpenAI u. a. *Solving Rubik's Cube with a Robot Hand*. 2019. arXiv: 1910.07113 [cs.LG].

[10] Gabriel Dulac-Arnold u. a. *An empirical investigation of the challenges of real-world reinforcement learning*. 2020. arXiv: 2003.11881 [cs.LG].

[11] Wenshuai Zhao, Jorge Peña Queralta und Tomi Westerlund. *Sim-to-Real Transfer in Deep Reinforcement Learning for Robotics: a Survey*. 2020. arXiv: 2009.13303 [cs.LG].

[12] Till Hörger u. a. „Preliminary Investigation of Robust Reinforcement Learning for Control of an Existing Green Propellant Thruster“. In: *2021 AIAA Propulsion and Energy Forum* (2021).

[13] Ernst Messerschmid und Stefanos Fasoulas. *Raumfahrtsysteme: Eine Einführung mit Übungen und Lösungen*. 4., neu bearb. Aufl. Berlin, Heidelberg: Springer-Verlag Berlin Heidelberg, 2011. ISBN: 978-3-642-12817-2. DOI: 10.1007/978-3-642-12817-2. URL: http://site.ebrary.com/lib/alltitles/docDetail.action?docID=10419670.

[14] G. P. Sutton und O. Biblarz. *Rocket Propulsion Elements*. New York: John Wiley & Sons, 2001. ISBN: 0-471-32642-9.

[15] Hydrazine REACH Autorisation Task Force of the European Space Industry. *Position Paper: Exemption of propellant related use of hydrazine from REACH authorisation requirement*. Hrsg. von ASD-Eurospace. URL: http://www.eurospace.org/Data/Sites/1/pdf/positionpapers/hydrazinereachpositionpaper_final_14june2012.pdf.

[16] Lukas Werling u. a. „Nitrous Oxide Fuels Blends: Research on premixed Monopropellants at the German Aerospace Center (DLR) since 2014". In: *AIAA Propulsion and Energy Forum 24.-26.08.2020*. 2020. URL: https://elib.dlr.de/135825/.

[17] Till Hörger. *Numerische und experimentelle Analyse der Wärmelasten auf eine N2O/C2H4 Green Propellant Brennkammer*. Deutsches Zentrum für Luft- und Raumfahrt, Institut für Raumfahrtantriebe, 2018.

[18] Brenden K. Petersen u. a. „Deep Reinforcement Learning and Simulation as a Path Toward Precision Medicine". In: *Journal of Computational Biology* 26.6 (2019). PMID: 30681362, S. 597–604. DOI: 10.1089/cmb.2018.0168. eprint: https://doi.org/10.1089/cmb.2018.0168. URL: https://doi.org/10.1089/cmb.2018.0168.

[19] Yuming Li, Pin Ni und Victor Chang. *Application of Deep Reinforcement Learning in tock Trading Strategies and Stock Forecasting*. 2019. DOI: 10.1007/s00607-019-00773-w.

[20] Volodymyr Mnih u. a. *Playing Atari with Deep Reinforcement Learning*. 2013. arXiv: 1312.5602 [cs.LG].

[21] Dong-Ok Won, Klaus-Robert Müller und Seong-Whan Lee. „An adaptive deep reinforcement learning framework enables curling robots with human-like performance in real-world conditions". In: *Science Robotics* 5.46 (2020). DOI: 10.1126/scirobotics.abb9764. eprint: https://robotics.sciencemag.org/content/5/46/eabb9764.full.pdf. URL: https://robotics.sciencemag.org/content/5/46/eabb9764.

[22] K. Kersandt, G. Muñoz und C. Barrado. „Self-training by Reinforcement Learning for Full-autonomous Drones of the Future*". In: *2018 IEEE/AIAA 37th Digital Avionics Systems Conference (DASC)*. 2018, S. 1–10. DOI: 10.1109/DASC.2018.8569503.

[23] Ameet V. Joshi. *Machine learning and artificial intelligence*. ISBN: 978-3-030-26621-9.

[24] Kai Dresia u. a. „Numerically Efficient Fatigue Life Prediction of Rocket Combustion Chambers using Artificial Neural Networks“. In: *Proceedings of the 8th European Conference for Aeronautics and Space Sciences*. 2019.

[25] Shanghang Zhang Hao Dong Zihan Ding. *Deep Reinforcement Learning: Fundamentals, Research and Applications*. SPRINGER NATURE, 30. Juni 2020. 514 S. ISBN: 9811540942. URL: https://www.ebook.de/de/product/38674753/deep_reinforcement_learning_fundamentals_research_and_applications.html.

[26] Richard S. Sutton und Andrew G. Barto. *Reinforcement learning: An introduction*. second edition. Bd. 2018: 1. Adaptive computation and machine learning. Cambridge, Mass.: The MIT Press, 2018. ISBN: 9780262039246.

[27] Mohit Sewak. *Deep Reinforcement Learning*. Springer Verlag, Singapore, 11. Juli 2019. 203 S. ISBN: 9811382840. URL: https://www.ebook.de/de/product/36524834/mohit_sewak_deep_reinforcement_learning.html.

[28] Richard Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[29] Josh Achiam. *Spinning Up*. Hrsg. von OpenAI Safety Team. 2018. URL: https://spinningup.openai.com/en/latest/index.html.

[30] Lukasz Kaiser u. a. *Model-Based Reinforcement Learning for Atari*. 2020. arXiv: 1903.00374 [cs.LG].

[31] Ashley Hill u. a. *Stable Baselines*. https://github.com/hill-a/stable-baselines. 2018.

[32] Christopher JCH Watkins und Peter Dayan. „Q-learning“. In: *Machine learning* 8.3-4 (1992), S. 279–292.

[33] Timothy P. Lillicrap u. a. *Continuous control with deep reinforcement learning*. 2019. arXiv: 1509.02971 [cs.LG].

[34] Scott Fujimoto, Herke van Hoof und David Meger. *Addressing Function Approximation Error in Actor-Critic Methods*. 2018. arXiv: 1802.09477 [cs.AI].

[35] Tuomas Haarnoja u. a. *Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor*. 2018.

[36] Ian Goodfellow, Yoshua Bengio und Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[37] Daoming Lyu u. a. „A Human-Centered Data-Driven Planner-Actor-Critic Architecture via Logic Programming“. In: *arXiv preprint arXiv:1909.09209* (2019).

[38] David Silver u. a. „Deterministic policy gradient algorithms“. In: 2014.

[39] OpenAI. *OpenAI Spinning Up: Soft Actor-Critic*. URL: https://spinningup.openai.com/en/latest/algorithms/sac.html.

[40] Lilian Weng. „Domain Randomization for Sim2Real Transfer“. In: *lilianweng.github.io/lil-log* (2019). URL: http://lilianweng.github.io/lil-log/2019/05/04/domain-randomization.html.

[41] Tianhong Dai u. a. *Analysing Deep Reinforcement Learning Agents Trained with Domain Randomisation*. 2020. arXiv: 1912.08324 [cs.LG].

[42] Yevgen Chebotar u. a. *Closing the Sim-to-Real Loop: Adapting Simulation Randomization with Real World Experience*. 2019. arXiv: 1810.05687 [cs.RO].

[43] Bhairav Mehta u. a. *Active Domain Randomization*. 2019. arXiv: 1904.04762 [cs.LG].

[44] Aayush Prakash u. a. *Structured Domain Randomization: Bridging the Reality Gap by Context-Aware Synthetic Data*. 2020. arXiv: 1810.10093 [cs.CV].

[45] Fereshteh Sadeghi und Sergey Levine. *CAD2RL: Real Single-Image Flight without a Single Real Image*. 2017. arXiv: 1611.04201 [cs.LG].

[46] Empresarios Agrupados Internacional. *EcosimPro 6.2.0*.

[47] Julian Dobusch. *Entwicklung und Tests von 3D-gedruckten, regenerativ gekühlten Versuchsbrennkammern für Raketentreibstoffe aus N2O und C2H6*. Deutsches Zentrum für Luft- und Raumfahrt, Institut für Raumfahrtantriebe, 2021.

[48] Konstantin Manassis. „Inbetriebnahme einer regenerativ gekühlten 22N Brennkammer für N2O/C2H6 Mono- und Bipropellants sowie Analyse der Verbrennungseffizienz und der Wärmelasten in Heißgasversuchen“. Universität Stuttgart, Institut für Raumfahrtsysteme.

[49] Till Hörger u. a. „Development of a Test Infrastructure for a Neural Network Controlled Green Propellant Thruster“. In: *8th Space Propulsion Comference* (2022).

[50] mtech. „MPG 03 PR Proportionalventil“. In: *Datenblatt* ().

[51] Leopold Böswirth. *Technische Strömungslehre Lehr- und Übungsbuch ; mit 43 Tabellen*. Wiesbaden: Vieweg + Teubner, 2010. ISBN: 9783834805232.

[52] Herbert Sigloch. *Technische Fluidmechanik*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014. DOI: 10.1007/978-3-642-54292-3.

[53] Philipp Moritz u. a. *Ray: A Distributed Framework for Emerging AI Applications*. 2018. arXiv: 1712.05889 [cs.DC].

[54] Eric Liang u. a. *RLlib: Abstractions for Distributed Reinforcement Learning*. 2018. arXiv: 1712.09381 [cs.AI].

[55] Greg Brockman u. a. *OpenAI Gym*. 2016. arXiv: 1606.01540 [cs.LG].

[56] Martín Abadi u. a. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[57] Günther Waxenegger-Wilfing u. a. „Heat Transfer Prediction for Methane in Regenerative Cooling Channels with Neural Networks“. In: *Journal of Thermophysics and Heat Transfer* 34.2 (2020), S. 347–357.

A. Anhang

Tab. A1.: Parameter des SAC-Algorithmus für die durchgeführten Berechnungen

Parameter	Wert
num_workers	4
num_envs_per_worker	1
create_env_on_driver	False
batch_mode	truncate_episodes
gamma	0.99
lr	0.0001
model	MODEL_DEFAULTS
optimizer	-
horizon	None
soft_horizon	False
no_done_at_end	False
env	None
observation_space	None
action_space	None
env_config	-
remote_worker_envs	False
remote_worker_envs	False
env_task_fn	None
render_env	False
record_env	False
clip_rewards	None
normalize_actions	True
clip_actions	False
preprocessor_pref	deepmind
log_level	WARN
callbacks	DefaultCallbacks
ignore_worker_failures	False
log_sys_usage	True
fake_sampler	False
framework	tf
eager_tracing	False
eager_max_retraces	20

explore	True
exploration_config	-
type	StochasticSampling
evaluation_interval	10
evaluation_duration	10
evaluation_duration_unit	episodes
evaluation_parallel_to_training	False
in_evaluation	False
evaluation_config	explore: False
evaluation_num_workers	1
custom_eval_function	None
always_attach_evaluation_results	False
keep_per_episode_custom_metrics	False
sample_async	False
sample_collector	SimpleListCollector
observation_filter	NoFilter
synchronize_filters	True
tf_session_args	-
intra_op_parallelism_threads	2
inter_op_parallelism_threads	2
gpu_options	-
allow_growth	True
log_device_placement	False
device_count	-
CPU	1
allow_soft_placement	True
local_tf_session_args	-
intra_op_parallelism_threads	8
inter_op_parallelism_threads	8
compress_observations	False
metrics_episode_collection_timeout_s	180
metrics_num_episodes_for_smoothing	100
min_time_s_per_reporting	None
min_train_timesteps_per_reporting	None
min_sample_timesteps_per_reporting	None
seed	None
extra_python_environs_for_driver	-
extra_python_environs_for_worker	-
num_gpus	0
_fake_gpus	False
num_cpus_per_worker	1
num_gpus_per_worker	0
custom_resources_per_worker	-

num_cpus_for_driver	1
placement_strategy	PACK
input	sampler
input_config	-
actions_in_input_normalized	False
input_evaluation	[is, wis]
postprocess_inputs	False
shuffle_buffer_size	0
output	None
output_config	-
output_compress_columns	[obs new_obs]
output_max_file_size	64 * 1024 * 1024
multiagent	-
policies	-
policy_map_capacity	100
policy_map_cache	None
policy_mapping_fn	None
policies_to_train	None
observation_fn	None
replay_mode	independent
count_steps_by	env_steps
logger_config	None
_tf_policy_handles_more_than_one_loss	False
_disable_preprocessor_api	False
_disable_action_flattening	False
_disable_execution_plan_api	False
disable_env_checking	False
simple_optimizer	-1
monitor	-1
evaluation_num_episodes	-1
metrics_smoothing_episodes	-1
min_iter_time_s	-1
collect_metrics_timeout	-1
twin_q	True
use_state_preprocessor	-1
fcnet_hiddens	[256 256]
fcnet_activation	relu
post_fcnet_hiddens	-
post_fcnet_activation	None
custom_model	None
custom_model_config	-
clip_actions	False
tau	5e-3

initial_alpha	1.0
target_entropy	auto
n_step	1
timesteps_per_iteration	500
buffer_size	-1
replay_buffer_config	-
type	MultiAgentReplayBuffer
capacity	100000
store_buffer_in_checkpoints	False
prioritized_replay	False
prioritized_replay_alpha	0.6
prioritized_replay_beta	0.4
prioritized_replay_eps	1e-6
prioritized_replay_beta_annealing_timesteps	20000
final_prioritized_replay_beta	0.4
compress_observations	False
training_intensity	512
optimization	-
actor_learning_rate	3e-4
critic_learning_rate	3e-4
entropy_learning_rate	3e-4
grad_clip	None
learning_starts	1500
rollout_fragment_length	1
train_batch_size	128
target_network_update_freq	0
worker_side_prioritization	False
min_time_s_per_reporting	1
_deterministic_loss	False
_use_beta_distribution	False

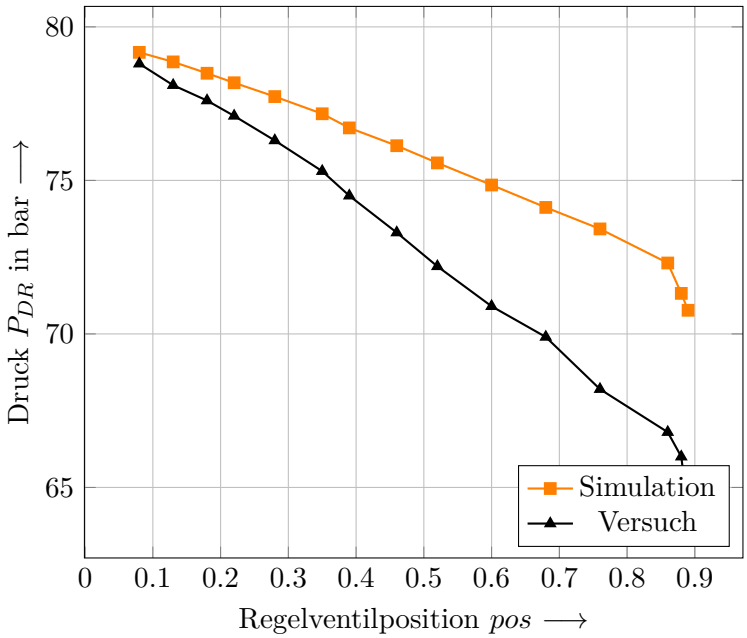


Abbildung A1.: Vergleich des Rohrdrucks hinter dem Druckregler von Simulation und Prüfstandsversuch

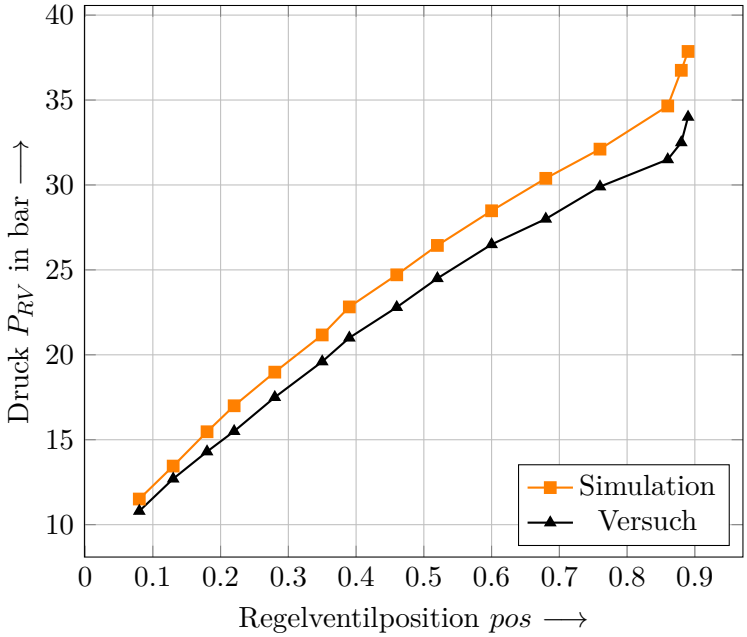


Abbildung A2.: Vergleich des Rohrdrucks hinter dem Regelventil von Simulation und Prüfstandsversuch

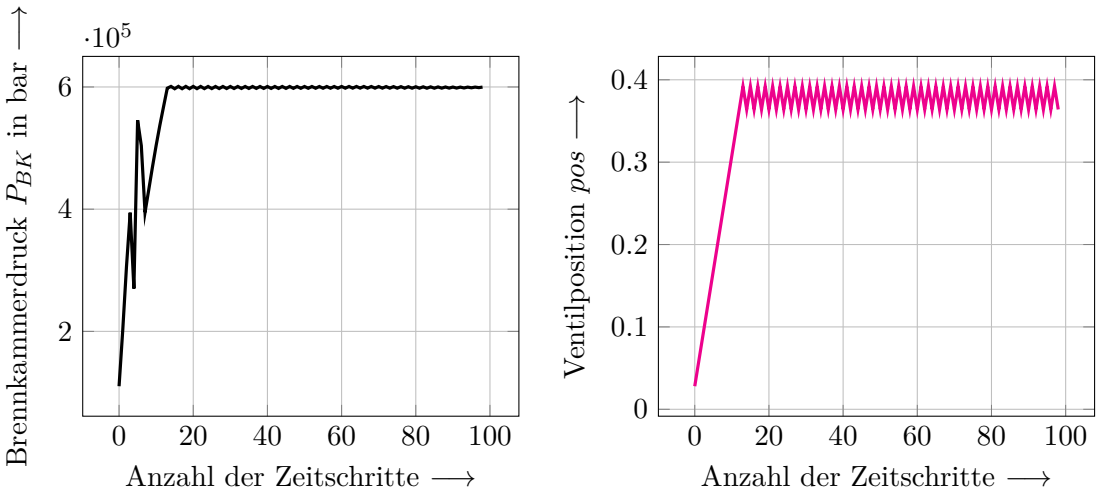


Abbildung A3.: Brennkammerdruck und Ventilposition der rudimentären Actionfunktion